Store trip analysis

by
Rui Furtado

Data analysis

Libraries

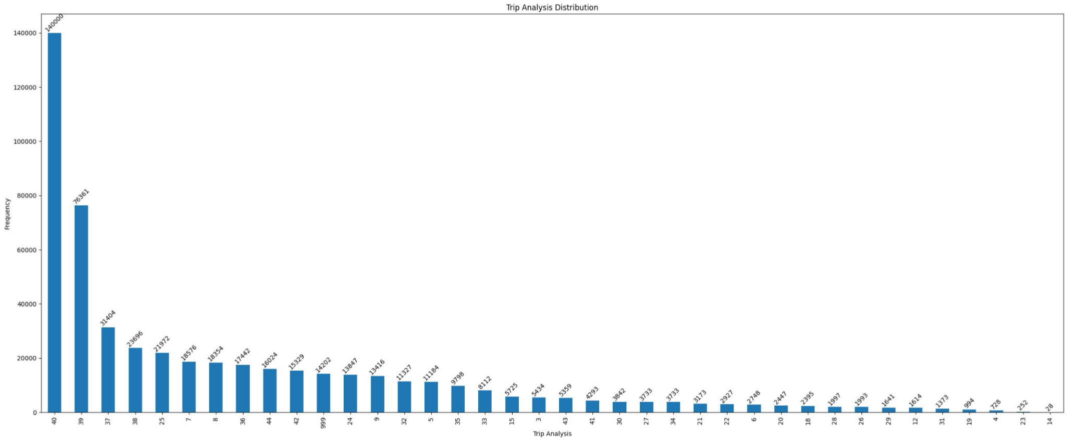
For developing this project the following libs were installed:

- fastparquet: parquet engine
- fastpivot: “shady lib” that creates a fast implementation of the pandas function pivot_table using scipy sparse matrices. It was great!
- matplotlib: for simple plotting
- pandas: data preprocessing
- pyarrow: apache arrow engine for saving parquet train data
- scikit-learn: ML algorithms
- uv: package manager

To activate the env use uv sync followed by source .venv/bin/activate or just pip install -r requirements if you don’t want to use a dedicated environment. The project uses python 3.12

TripType

- Target column. looks pretty unbalanced. We will need to tackle this issue when creating our prediction algorithm either with:
- using algorithms that deal with this issue
- over/under sample in order to have more/less examples
- 38 classes represented



VisitNumber

- Will be used as primary key
- 76605 elements
- No nulls
- Will be used as index for our data
- Multiple records by visit
- model will predict by store trip so records need to be aggregated by VisitNumber

Weekday

- String with weekday of record
- No nulls
- Values need to be correctly standardized - lower, trimmed
- Cyclical encoding was chosen
- Features can be derived from it - is_weekend

```
Weekday
Sunday       104555
Saturday      89807
Friday        72054
Monday        62828
Wednesday     57141
Tuesday       55902
Thursday      48186
SATURDAY       7000
THURSDAY       6000
FrIdAy         5000
MonDAY         4000
sunday         3000
tuesday        2000
Name: count, dtype: int64
```

Upc

- Product code
- Number of nans - 3331 - will be dropped
- High cardinality - 85895 elements
- No nulls
- Will be one-hot encoded to generate a column by product (aggregated by visit)

ScanCount

- Product code
- High cardinality - 85895 elements
- No nulls
- Will be one-hot encoded to generate a column by product (aggregated by visit)
- Presents negative numbers (from -1 to -10). These were kept since they were returns
- Used to create total count of products (after agg)

DepartmentDescription

- Product categorization describe department
- 99 elements
- Should be cleaned - lowercase, trim, space removal
- After cleaning we got 67 elements
- Will be also one-hot encoded to generate 1 column by department (agg by VisitNumber)

FineLineNumber

- Another categorization of the product
- Presented 3331 nulls
- Was dropped since we wanted to use UPC for column dummies
- Should have used this one since cardinality was lower (5126 elements vs 85895 elements)

Preprocessing

Encoding of text categories

- Used regex for removing non-chars, trimmed and lowered
- Applied to weekday and department description

```python
# Function for standardization
def standardize_text_categories(cat:str) -> str:
    """Standardizes given text

    Args:
        cat (str): category to be standardize

    Returns:
        standardize text
    """
    return re.sub(r'[^a-zA-Z]', '', cat.strip().lower()) # remove non-letters
```

Aggregation by visit and total

- Aggregation by visit number and cols that identify a visit - groupby VisitNumber, TripType, Weekday
- Sum of ScanCount to create Total - total number of products acquired per visit
- Records reduced from 517475 to 75456

```
# Remove rows with no Upc
data_cleaned = data_cleaned.dropna(subset=['Upc'])

visit_agg = data_cleaned[['VisitNumber','TripType','Weekday','ScanCount']].groupby(['VisitNumber','TripType','Weekday'],as_index=False).sum()
visit_agg=visit_agg.set_index('VisitNumber').rename(columns={"ScanCount":"Total"}) # set index to VisitNumber to be able to join and rename ScanCount
visit_agg
```

Pivot from UPC, visitNumber and ScanCount

- Creation of Pivot table for getting a dataframe with visit numbers by number of products acquired by product type
- Usage of fastpivot lib to address memory issues caused by high cardinality - sparse matrix creation
- Inner join with aggregated visit table by index (VisitNumber) - 74566 x 85894 matrix

```python
from fastpivot import pivot_sparse

pivot_upc=pivot_sparse(data_cleaned, index='VisitNumber', columns='Upc', values='ScanCount', fill_value=0).fillna(0)
pivot_upc
```

✓ 4.7s Python

Upc	8.340000e+02	3.032000e+03	3.035000e+03	3.066000e+03	3.082000e+03	3.100000e+03	3.107000e+03	3.112000e+03	3.121000e+03	3.127000e+03	...	9.781
VisitNumber												
8	0	0	0	0	0	0	0	0	0	0	...	
11	0	0	0	0	0	0	0	0	0	0	...	
12	0	0	0	0	0	0	0	0	0	0	...	
15	0	0	0	0	0	0	0	0	0	0	...	
19	0	0	0	0	0	0	0	0	0	0	...	
...	...	...	...	...	...	...	...	...	...	...	...	
191335	0	0	0	0	0	0	0	0	0	0	...	
191343	0	0	0	0	0	0	0	0	0	0	...	
191344	0	0	0	0	0	0	0	0	0	0	...	
191346	0	0	0	0	0	0	0	0	0	0	...	
191347	0	0	0	0	0	0	0	0	0	0	...	

75456 rows × 85894 columns

Departement description dummies

- One-hot encoding of department description and aggregation by visit number
- Group by VisiNumber to get frequency of departments by visit
- Usage of sparse=True in pd.get_dummies to use sparse matrix
- Joined with previous aggregated matrix by VisitNumber

```python
department_dummies = pd.get_dummies(
    data_cleaned.set_index('VisitNumber').DepartmentDescription,dtype=int,sparse=True
).groupby('VisitNumber').sum()
department_dummies
```

✓ 1m 27.0s Python

itNumber	accessories	automotive	bakery	bathandshower	beauty	bedding	booksandmagazines	boyswear	brasshapewear	camerasandsupplies	...	seafood	seasonal	servicedeli
8	0	0	0	0	0	0	0	0	0	0	...	0	0	0
11	0	0	0	0	0	0	0	0	0	0	...	0	0	0
12	0	0	0	0	0	0	0	2	0	0	...	0	0	0
15	0	0	0	0	0	0	0	0	0	0	...	0	0	0
19	1	0	0	0	0	0	0	0	0	0	...	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
191335	0	0	0	0	0	0	0	0	0	0	...	0	0	0
191343	0	0	0	0	0	0	0	0	0	0	...	0	0	0
191344	0	0	0	0	4	0	0	0	0	0	...	0	0	0
191346	0	0	0	0	0	0	0	0	0	0	...	0	0	0
191347	0	0	0	0	0	0	0	0	0	0	...	0	0	0

6 rows × 67 columns

Weekday encoding

- Cyclical encoding with cosine and sine to preserve proximity of week final with week beginning
- Creation of is_weekend
- Creation of final dataframe - 75456 x 85966

```python
DAY_MAPPING= {
    'monday': 1, 'tuesday': 2, 'wednesday': 3, 'thursday': 4,
    'friday': 5, 'saturday': 6, 'sunday': 7
}

day_num = agg['Weekday'].map(DAY_MAPPING)

# Encode weekday into cyclical features
agg['weekday_sin'] = np.sin((2 * np.pi / 7)* day_num)
agg['weekday_cos'] = np.cos((2 * np.pi / 7)* day_num)

# check when we have a weekend
agg['is_weekend'] = agg['Weekday'].isin(['saturday', 'sunday']).astype(int)
agg = agg.drop('Weekday',axis=1)
agg
```
✓ 0.4s

Data storage

- Data stored in parquet - to_csv too slow for high volume of data
- Needed to convert pandas types from sparse to int and float
- High memory usage! Should have used FineLineNumber instead of Upc for one-hot encoding

```
type_dict = {**dict.fromkeys(agg.columns, int), "weekday_sin": float, "weekday_cos": float}
agg.info(memory_usage='deep')
agg = agg.astype(type_dict)
agg.info(memory_usage='deep')
 8.2s

<class 'pandas.DataFrame'>
Index: 75456 entries, 8 to 191347
Columns: 85966 entries, TripType to is_weekend
dtypes: Sparse[int64, 0](85961), float64(2), int64(3)
memory usage: 12.1 MB
<class 'pandas.DataFrame'>
Index: 75456 entries, 8 to 191347
Columns: 85966 entries, TripType to is_weekend
dtypes: float64(2), int64(85964)
memory usage: 48.3 GB

agg.to_parquet('processed_data.parquet')
```

Modeling

Data splitting

- Data was split into train and test - 80% train and 20% for test
- Stratified to have a test set that resembles class distribution of train - since we have imbalance of classes this is needed in order to have all classes present

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=4,
    stratify=y,
    shuffle=True
)
```

✓ 1m 22.1s

Baseline model

- Random forest model was chosen for baseline prediction:
  - No need for feature scaling
  - Resistant to overfitting
  - Only looks at a subset of features at a time
  - Handles non-linear relationships
  - Provides feature importance
  - Provides balancing option off the shelf
- Model took around 40 min to train due to high cardinality ! (Mistake - trained with 500 trees and full data. train and test results were too similar. After correction 9min (200 trees and corrected data)
- Model was immediately saved to pkl file after training to avoid re-fitting

Baseline model

- Parameters chosen
  - 200 trees
  - Maximum depth of each tree 25 branches to try to prevent overffiting
  - class_weight=balanced to penalize rare class misclassification

```python
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(
    n_estimators=200,
    max_depth=20,
    random_state=43,
    class_weight='balanced',
    n_jobs=-1,
).fit(X_train.values,y_train.values)
```

✓ 9m 6.8s

Results train prediction

- Results were not great even for prediction with X_train
- Model is underfiiting
- Some classes still present good results
- F1-score of 51% - same as tossing a coin
- More predictive power maybe would Be helpfull

	precision	recall	f1-score	support
3.0	0.94	0.93	0.94	2327
4.0	0.10	0.96	0.19	226
5.0	0.88	0.14	0.24	2181
6.0	0.40	0.93	0.56	808
7.0	0.63	0.57	0.60	3694
8.0	0.82	0.36	0.50	7826
9.0	0.97	0.01	0.02	6028
12.0	0.67	0.91	0.77	163
14.0	0.05	1.00	0.10	3
15.0	0.39	0.58	0.47	622
18.0	0.32	0.85	0.47	354
19.0	0.24	0.97	0.39	245
20.0	0.29	0.99	0.45	407
21.0	0.30	0.90	0.45	418
22.0	0.61	0.35	0.45	600
23.0	0.14	1.00	0.25	84
24.0	0.49	0.74	0.59	1662
25.0	0.49	0.71	0.58	2374
26.0	0.25	0.94	0.39	317
27.0	0.36	0.93	0.52	508
28.0	0.27	0.99	0.42	295
29.0	0.97	0.41	0.58	277
30.0	0.32	0.84	0.46	682
31.0	0.56	1.00	0.72	369
32.0	0.51	0.93	0.66	1280
33.0	0.38	0.74	0.50	858
34.0	0.33	0.96	0.49	454
35.0	0.42	0.47	0.44	1274
36.0	0.42	0.63	0.51	1917
37.0	0.37	0.86	0.52	1797
38.0	0.29	0.56	0.39	1851
39.0	0.94	0.02	0.04	6343
40.0	0.65	0.89	0.75	3971
41.0	0.98	0.45	0.62	370
42.0	0.99	0.22	0.36	1167
43.0	1.00	0.07	0.13	579
44.0	0.90	0.59	0.72	746
999.0	1.00	0.81	0.90	5287
accuracy			0.51	60364
macro avg	0.54	0.69	0.48	60364
weighted avg	0.72	0.51	0.47	60364

Results test prediction

- Results were also not good for testset prediction
- F1-score of 46%

	precision	recall	f1-score	support
3.0	0.95	0.92	0.93	582
4.0	0.09	0.80	0.16	56
5.0	0.65	0.20	0.30	545
6.0	0.33	0.90	0.49	202
7.0	0.60	0.56	0.58	924
8.0	0.78	0.35	0.49	1957
9.0	0.94	0.09	0.16	1507
12.0	0.20	0.38	0.26	40
14.0	0.00	0.00	0.00	0
15.0	0.26	0.54	0.35	156
18.0	0.23	0.65	0.34	88
19.0	0.25	0.90	0.39	61
20.0	0.29	0.85	0.43	102
21.0	0.26	0.78	0.39	105
22.0	0.41	0.29	0.34	150
23.0	0.17	0.95	0.29	21
24.0	0.47	0.64	0.55	415
25.0	0.52	0.67	0.59	594
26.0	0.20	0.95	0.32	79
27.0	0.33	0.80	0.47	127
28.0	0.21	0.84	0.33	74
29.0	0.32	0.10	0.15	69
30.0	0.28	0.55	0.37	170
31.0	0.53	0.98	0.68	92
32.0	0.50	0.87	0.63	320
33.0	0.26	0.74	0.38	215
34.0	0.30	0.89	0.45	114
35.0	0.34	0.35	0.35	319
36.0	0.44	0.40	0.42	479
37.0	0.30	0.77	0.43	449
38.0	0.23	0.39	0.29	463
39.0	0.53	0.01	0.01	1586
40.0	0.51	0.77	0.61	993
41.0	0.00	0.00	0.00	92
42.0	0.46	0.02	0.04	292
43.0	0.00	0.00	0.00	145
44.0	0.37	0.07	0.12	187
999.0	1.00	0.79	0.88	1322
accuracy			0.46	15092
macro avg	0.38	0.55	0.37	15092
weighted avg	0.60	0.46	0.43	15092

Model #2 No Upc

- Since training took some time, I decided based on RF feature importance, to remove all the features related with Upc

	Feature	Importance
0	Total	0.028035
85945	petsandsupplies	0.022452
85948	playersandelectronics	0.017128
85961	wireless	0.016792
85896	automotive	0.016388
85912	electronics	0.015778
85951	produce	0.015512
85958	sportinggoods	0.015224
85914	financialservices	0.014589
85929	infantconsumablehardlines	0.014288
85773	692303000000.0	0.013809
85934	lawnandgarden	0.013122
60699	68113102889.0	0.012862
85927	impulsemerchandise	0.012508
85960	toys	0.012196
85913	fabricsandcrafts	0.012036
85939	officesupplies	0.011325
57899	64541689243.0	0.011053
85906	celebration	0.010997
85956	shoes	0.010514

Model #2 No Upc - Results train prediction

- Results improved 91% for train
- Model had the same

params but max_depth was set

to None

	precision	recall	f1-score	support
3.0	0.81	0.98	0.89	2158
4.0	0.25	0.94	0.40	203
5.0	0.91	0.74	0.81	2067
6.0	0.84	0.97	0.90	750
7.0	0.91	0.95	0.93	3449
8.0	0.91	0.86	0.89	7364
9.0	0.95	0.66	0.78	5684
12.0	0.82	0.99	0.90	164
14.0	1.00	1.00	1.00	2
15.0	0.90	0.95	0.92	594
18.0	0.77	0.94	0.84	332
19.0	0.41	0.99	0.58	233
20.0	0.72	0.98	0.83	386
21.0	0.88	0.98	0.93	373
22.0	0.84	0.78	0.81	539
23.0	0.42	1.00	0.59	75
24.0	0.90	0.98	0.94	1537
25.0	0.96	0.98	0.97	2246
26.0	0.73	0.99	0.84	289
27.0	0.87	0.98	0.92	481
28.0	0.64	0.98	0.78	278
29.0	0.60	0.94	0.74	256
30.0	0.68	0.99	0.81	650
31.0	0.68	0.99	0.81	341
32.0	0.89	1.00	0.94	1185
33.0	0.98	0.99	0.99	820
34.0	0.88	0.99	0.93	408
35.0	0.97	0.97	0.97	1205
36.0	0.97	0.98	0.97	1820
37.0	0.94	0.98	0.96	1708
38.0	0.97	0.99	0.98	1702
39.0	1.00	0.99	1.00	5964
40.0	1.00	1.00	1.00	3674
41.0	1.00	1.00	1.00	341
42.0	1.00	0.99	0.99	1088
43.0	1.00	1.00	1.00	570
44.0	1.00	1.00	1.00	703
999.0	0.99	0.83	0.90	4953
accuracy			0.91	56592
macro avg	0.84	0.95	0.88	56592
weighted avg	0.93	0.91	0.91	56592

Model #2 No Upc - Results test prediction

- Test set results improved to 64% F1-score
- Still with problems in

minority classes

	precision	recall	f1-score	support
3.0	0.78	0.98	0.87	751
4.0	0.17	0.49	0.25	79
5.0	0.58	0.53	0.55	659
6.0	0.66	0.74	0.70	260
7.0	0.67	0.65	0.66	1169
8.0	0.78	0.77	0.77	2419
9.0	0.72	0.52	0.60	1851
12.0	0.07	0.05	0.06	39
14.0	0.00	0.00	0.00	1
15.0	0.46	0.48	0.47	184
18.0	0.29	0.47	0.36	110
19.0	0.20	0.55	0.30	73
20.0	0.46	0.77	0.58	123
21.0	0.56	0.68	0.62	150
22.0	0.42	0.35	0.38	211
23.0	0.32	0.50	0.39	30
24.0	0.57	0.49	0.53	540
25.0	0.64	0.69	0.66	722
26.0	0.38	0.47	0.42	107
27.0	0.46	0.65	0.54	154
28.0	0.33	0.62	0.43	91
29.0	0.14	0.17	0.15	90
30.0	0.39	0.56	0.46	202
31.0	0.52	0.90	0.66	120
32.0	0.65	0.85	0.73	415
33.0	0.62	0.58	0.60	253
34.0	0.57	0.61	0.59	160
35.0	0.57	0.56	0.57	388
36.0	0.61	0.63	0.62	576
37.0	0.56	0.48	0.52	538
38.0	0.54	0.32	0.40	612
39.0	0.51	0.69	0.59	1965
40.0	0.76	0.91	0.83	1290
41.0	0.54	0.06	0.10	121
42.0	0.39	0.16	0.23	371
43.0	0.19	0.02	0.04	154
44.0	0.31	0.03	0.06	230
999.0	0.96	0.72	0.82	1656
accuracy			0.64	18864
macro avg	0.48	0.52	0.48	18864
weighted avg	0.65	0.64	0.63	18864

Model #3 SMOTE - Results train prediction

- For model 3 we decided to use SMOTE with no Upc to generate more samples and try to overcome issues with minority classes
- Max_depth was set to

30 this time

	precision	recall	f1-score	support
3.0	0.91	0.96	0.93	7364
4.0	0.76	1.00	0.86	7364
5.0	0.91	0.65	0.76	7364
6.0	0.95	0.96	0.95	7364
7.0	0.90	0.76	0.83	7364
8.0	0.56	0.83	0.67	7364
9.0	0.88	0.41	0.56	7364
12.0	1.00	0.99	0.99	7364
14.0	1.00	1.00	1.00	7364
15.0	0.95	0.82	0.88	7364
18.0	0.93	0.90	0.92	7364
19.0	0.80	1.00	0.89	7364
20.0	0.95	1.00	0.97	7364
21.0	0.96	0.97	0.97	7364
22.0	0.97	0.78	0.87	7364
23.0	0.98	1.00	0.99	7364
24.0	0.87	0.94	0.90	7364
25.0	0.89	0.88	0.89	7364
26.0	0.93	1.00	0.96	7364
27.0	0.97	1.00	0.98	7364
28.0	0.91	1.00	0.95	7364
29.0	0.95	0.92	0.93	7364
30.0	0.89	0.96	0.92	7364
31.0	0.96	0.96	0.96	7364
32.0	0.95	0.98	0.96	7364
33.0	0.94	0.97	0.95	7364
34.0	0.96	1.00	0.98	7364
35.0	0.74	0.90	0.81	7364
36.0	0.89	0.91	0.90	7364
37.0	0.91	0.94	0.92	7364
38.0	0.83	0.93	0.87	7364
39.0	0.99	0.76	0.86	7364
40.0	1.00	0.99	0.99	7364
41.0	1.00	0.99	1.00	7364
42.0	1.00	0.90	0.95	7364
43.0	1.00	0.96	0.98	7364
44.0	1.00	0.99	0.99	7364
999.0	1.00	0.76	0.86	7364
accuracy			0.91	279832
macro avg	0.92	0.91	0.91	279832
weighted avg	0.92	0.91	0.91	279832

Model #3 SMOTE - Results test prediction

- Despite balancing classes result did not improve
- F1-score 61%

	precision	recall	f1-score	support
3.0	0.78	0.96	0.86	751
4.0	0.16	0.59	0.25	79
5.0	0.54	0.50	0.52	659
6.0	0.56	0.82	0.67	260
7.0	0.74	0.57	0.65	1169
8.0	0.74	0.80	0.77	2419
9.0	0.82	0.33	0.47	1851
12.0	0.10	0.05	0.07	39
14.0	0.00	0.00	0.00	1
15.0	0.48	0.52	0.50	184
18.0	0.29	0.64	0.40	110
19.0	0.22	0.70	0.33	73
20.0	0.35	0.91	0.51	123
21.0	0.50	0.74	0.60	150
22.0	0.41	0.37	0.39	211
23.0	0.33	0.60	0.42	30
24.0	0.57	0.53	0.55	540
25.0	0.63	0.62	0.62	722
26.0	0.31	0.67	0.43	107
27.0	0.44	0.75	0.55	154
28.0	0.27	0.82	0.41	91
29.0	0.15	0.16	0.15	90
30.0	0.33	0.66	0.44	202
31.0	0.53	0.92	0.67	120
32.0	0.59	0.87	0.71	415
33.0	0.49	0.68	0.57	253
34.0	0.44	0.78	0.56	160
35.0	0.47	0.62	0.53	388
36.0	0.57	0.65	0.61	576
37.0	0.53	0.57	0.55	538
38.0	0.40	0.45	0.42	612
39.0	0.60	0.49	0.54	1965
40.0	0.78	0.91	0.84	1290
41.0	0.33	0.05	0.09	121
42.0	0.42	0.17	0.24	371
43.0	0.25	0.03	0.06	154
44.0	0.33	0.07	0.11	230
999.0	1.00	0.71	0.83	1656
accuracy			0.61	18864
macro avg	0.46	0.56	0.47	18864
weighted avg	0.65	0.61	0.61	18864

Model #4 K best

- Based on the previous features importance lets explore using only a predefined number of features k=1000. This will include Upc and the rest

	Feature	Importance
0	Total	0.028035
85945	petsandsupplies	0.022452
85948	playersandelectronics	0.017128
85961	wireless	0.016792
85896	automotive	0.016388
85912	electronics	0.015778
85951	produce	0.015512
85958	sportinggoods	0.015224
85914	financialservices	0.014589
85929	infantconsumablehardlines	0.014288
85773	692303000000.0	0.013809
85934	lawnandgarden	0.013122
60699	68113102889.0	0.012862
85927	impulsemerchandise	0.012508
85960	toys	0.012196
85913	fabricsandcrafts	0.012036
85939	officesupplies	0.011325
57899	64541689243.0	0.011053
85906	celebration	0.010997
85956	shoes	0.010514

Model #4 K best - Results train prediction

- Training improved a bit 94%. This might represent overfitting

	precision	recall	f1-score	support
3.0	0.97	1.00	0.98	2180
4.0	0.41	0.87	0.56	210
5.0	0.90	0.87	0.88	2032
6.0	0.88	0.97	0.92	733
7.0	0.94	0.97	0.96	3489
8.0	0.93	0.91	0.92	7414
9.0	0.96	0.71	0.82	5674
12.0	0.99	1.00	1.00	152
14.0	1.00	1.00	1.00	2
15.0	0.94	0.96	0.95	572
18.0	0.82	0.96	0.88	341
19.0	0.62	0.93	0.74	222
20.0	0.83	0.97	0.89	385
21.0	0.90	0.99	0.95	400
22.0	0.74	0.91	0.82	572
23.0	0.65	1.00	0.79	77
24.0	0.89	0.99	0.94	1581
25.0	0.96	0.99	0.97	2204
26.0	0.76	0.99	0.86	292
27.0	0.90	0.99	0.94	466
28.0	0.66	1.00	0.80	275
29.0	0.67	0.97	0.79	262
30.0	0.68	0.99	0.81	618
31.0	0.77	1.00	0.87	344
32.0	0.90	1.00	0.95	1216
33.0	0.99	0.99	0.99	813
34.0	0.90	0.99	0.94	416
35.0	0.98	0.99	0.98	1202
36.0	0.97	0.97	0.97	1781
37.0	1.00	1.00	1.00	1702
38.0	0.99	0.99	0.99	1744
39.0	1.00	1.00	1.00	5943
40.0	1.00	1.00	1.00	3685
41.0	0.99	1.00	1.00	351
42.0	1.00	0.99	1.00	1081
43.0	1.00	1.00	1.00	530
44.0	1.00	1.00	1.00	720
999.0	0.99	0.92	0.95	4911
accuracy			0.94	56592
macro avg	0.88	0.97	0.92	56592
weighted avg	0.95	0.94	0.94	56592

Model #4 K best - Results test prediction

- Results slightly improve but we still have the same issues as before
- F1-score of 65%, maybe due to better score on test?
- Would need to cross-validate to better understand

	precision	recall	f1-score	support
3.0	0.94	0.98	0.96	729
4.0	0.17	0.26	0.21	72
5.0	0.63	0.64	0.63	694
6.0	0.72	0.70	0.71	277
7.0	0.66	0.69	0.68	1129
8.0	0.79	0.79	0.79	2369
9.0	0.69	0.56	0.62	1861
12.0	0.22	0.04	0.07	51
14.0	0.00	0.00	0.00	1
15.0	0.51	0.37	0.43	206
18.0	0.32	0.50	0.39	101
19.0	0.29	0.33	0.31	84
20.0	0.49	0.66	0.56	124
21.0	0.55	0.63	0.58	123
22.0	0.39	0.53	0.45	178
23.0	0.35	0.32	0.33	28
24.0	0.56	0.60	0.58	496
25.0	0.66	0.68	0.67	764
26.0	0.36	0.41	0.39	104
27.0	0.49	0.73	0.59	169
28.0	0.30	0.44	0.36	94
29.0	0.23	0.23	0.23	84
30.0	0.38	0.44	0.41	234
31.0	0.60	0.89	0.72	117
32.0	0.60	0.79	0.68	384
33.0	0.60	0.52	0.56	260
34.0	0.58	0.62	0.60	152
35.0	0.57	0.57	0.57	391
36.0	0.59	0.61	0.60	615
37.0	0.59	0.45	0.51	544
38.0	0.54	0.34	0.41	570
39.0	0.50	0.71	0.59	1986
40.0	0.75	0.87	0.81	1279
41.0	0.50	0.05	0.10	111
42.0	0.48	0.18	0.26	378
43.0	0.11	0.01	0.01	194
44.0	0.60	0.03	0.05	213
999.0	0.96	0.83	0.89	1698
accuracy			0.65	18864
macro avg	0.51	0.50	0.48	18864
weighted avg	0.66	0.65	0.64	18864

TODOS

This is challenge is not completed and due to time constraints I was not able to fully implement all the strategies that I had in my mind for reaching a better outcome. Some steps I would take to improve the presented results:

- Explore the usage of FineLineNumber instead of Upc. Upc columns could be discarded since they were barely used by the final model
- Employ a cross validation strategy to access results- Current results are not 100% trustworthy since this was not applied. The stratified version should be used since this is a multiclass problem.
- Use a gridsearch to explore better hyperparameters for the random forest.
- Explore other models such as GradientBoostingTress or XGboost
- Explore other strategies of feature selection for example having k-best in GridSearch
- Explore using bins for some of the features - for example aggregating similar departments or even removing or join of TripTypes