Transit Demand Forecast

By Firoz Kabir (208448805)
For ITEC 6970 - Winter 2025

Introduction

In this final project, I will be using Data Centric Machine Learning techniques learned in the course to forecast demand for Toronto Island ferries based on historical data. The intent of this project is to create a transit demand forecast pipeline that can be generalized to any transit system. However, due to lack of data for other modes of transit in Toronto, this assumption of generalization could not be tested or adjusted for.

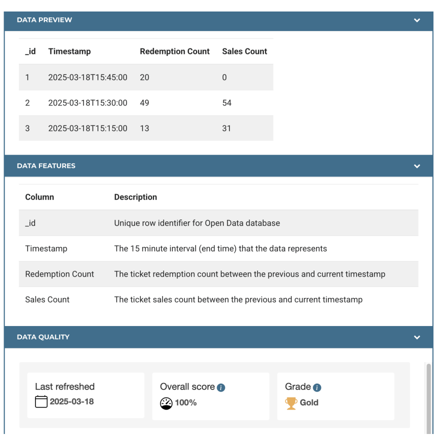
Related Work

The work presented in this project report was heavily influenced by Rob Mulla's work on Time Series Forecasting using XGBoost on Kaggle [1]. In his work, Rob demonstrates the importance of Exploratory Data Analysis (EDA), Time Series Split, Lag Features and Cross Validation in improving the quality of a forecast model based.

In this course, I also learned the importance of above techniques in a data centric machine learning application. In this course project, I have tried to apply these techniques to a real world application of forecasting transit demand in Toronto using the data available to us publicly.

Data Collection

Toronto Island Ferry Ticket Counts data is available through the Toronto Open Data portal [2].


DATA PREVIEW

_id	Timestamp	Redemption Count	Sales Count
1	2025-03-18T15:45:00	20	0
2	2025-03-18T15:30:00	49	54
3	2025-03-18T15:15:00	13	31

DATA FEATURES

Column	Description
_id	Unique row identifier for Open Data database
Timestamp	The 15 minute interval (end time) that the data represents
Redemption Count	The ticket redemption count between the previous and current timestamp
Sales Count	The ticket sales count between the previous and current timestamp

DATA QUALITY

Last refreshed
2025-03-18

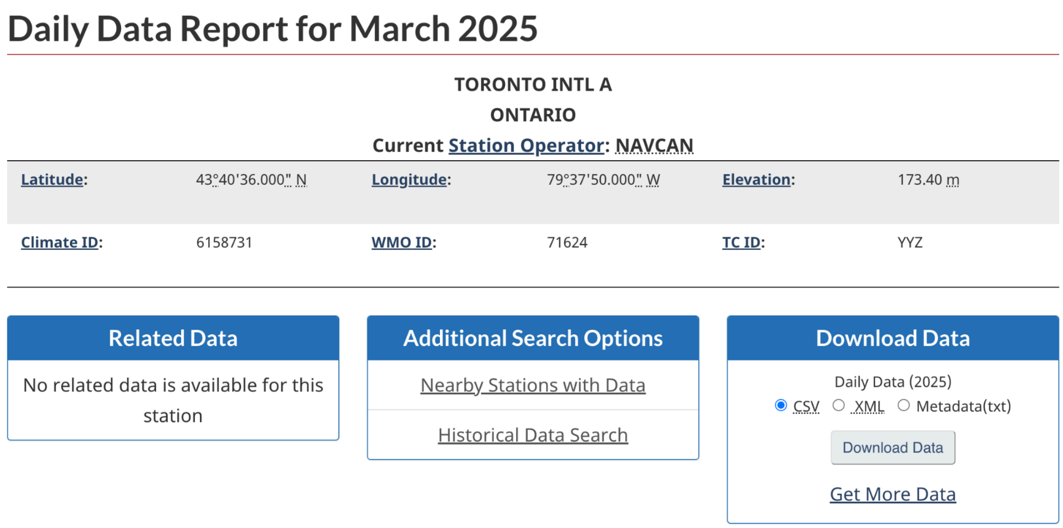
Overall score
100%

Grade
Gold

This dataset includes the features _id, Timestamp, Redemption Count and Sales Count. **Timestamp** is in 15 minute intervals (end time) that the data represents. **Redemption Count** is the ticket redemption count between the previous and current timestamp. **Sales Count** is the ticket sales count between the previous and current timestamp.

Timestamp and Redemption Count were used as the core time series dataset for this forecasting model.

Historical weather data is available through the Government of Canada's portal [3].
While the dataset contains a lot of attributes, we are only using the temperature in our model.
A later section will elaborate the rationale for that.



In addition, a python package "holidays" was used to identify dates that are holidays in Canada. This is also part of the dataset used in the model. More details on this will follow in the section describing the model.

Finally, the notebooks for this project include a section to download the dataset from the source as needed. A snippet of the code is shown below.

Download files to local if running or the first time

```
download_files = False
```

```
def download_weather_data(start_year=2016,
                          end_year=2024,
                          station_id=51459):

    local_path = f".{os.sep}data{os.sep}weather-data{os.sep}"
    print(f"Weather data will be saved in {local_path}")
    Path(local_path).mkdir(parents=True, exist_ok=True)

    for year in range(start_year, end_year+1):

        for month in range(12):
            url = f"https://climate.weather.gc.ca/climate_data/bulk_data_e.html?format=csv&stationID=51459&Year={year}&Month={month+1}&Day=1&timeframe=1&submit= Download+Data"
            file = f"{local_path}{os.sep}{year}_{month+1}.csv"
            print(f"Downloading weather data for year: {year}, month: {month+1} into file: {file}")
            response = r.get(url)
            df = pd.read_csv(io.StringIO(response.content.decode('utf-8')))
            df.to_csv(file, index=False, header=True, quoting=csv.QUOTE_ALL)

    print("== done ==")
```

```
def download_ferry_ticket_data():
    url = "https://ckan0.cf.opendata.inter.prod-toronto.ca/dataset/toronto-island-ferry-ticket-counts/resource/c46719f5-8006-44e1-8b1e-5ad90bb9f6f4/download/Toronto%20Island%20Ferry%20
    file = f".{os.sep}data{os.sep}toronto_island_ferry_ticket_counts.csv"
    print(f"Downloading Toronto ferry traffic data from Toronto Open Data portal into file: {file}")
    response = r.get(url)
    df = pd.read_csv(io.StringIO(response.content.decode('utf-8')))
    df.to_csv(file, index=False, header=True, quoting=csv.QUOTE_ALL)
    print("== done ==")
```

```
if download_files:
    download_weather_data()
    download_ferry_ticket_data()
```

Exploratory Data Analysis (EDA)

While loading the data, I declared timestamp as the index. The Exploratory Data Analysis started with running df.info() to confirm there are not nulls in the redemption_count column.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 240821 entries, 2025-03-21 08:45:00 to 2015-05-01 13:30:00
Data columns (total 1 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   redemption_count  240821 non-null  int64
dtypes: int64(1)
memory usage: 3.7 MB
```

A df.head() call shows we have data every 15 minutes and timestamps are processed as index.

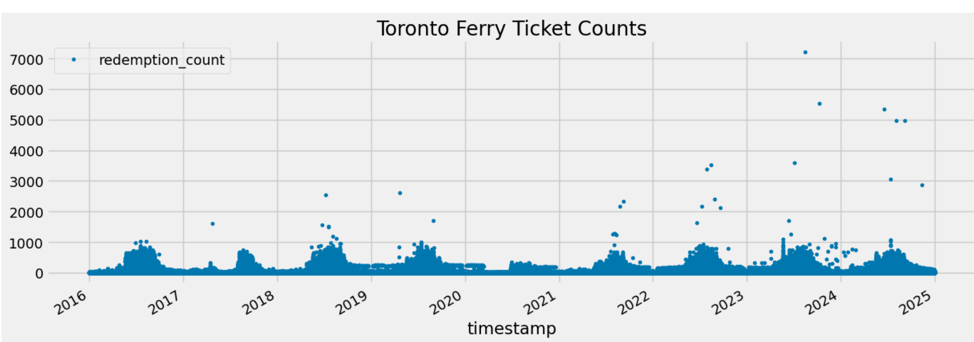
```
df.head()
```

	redemption_count
timestamp	
2025-03-21 08:45:00	2
2025-03-21 08:30:00	9
2025-03-21 08:15:00	32
2025-03-21 08:00:00	6
2025-03-21 07:45:00	0

```
df.describe()
```

	redemption_count
count	220937.000000
mean	46.562187
std	101.554887
min	0.000000
25%	3.000000
50%	10.000000
75%	37.000000
max	7216.000000

Furthermore, df.describe() provided an ANOVA table. From the mean, standard deviation and range (min, max) it is clearly visible there may be some outliers.
Next, a look at a plot of the dataset shows there are some sparse outliers, specifically in years 2023 - 2025.



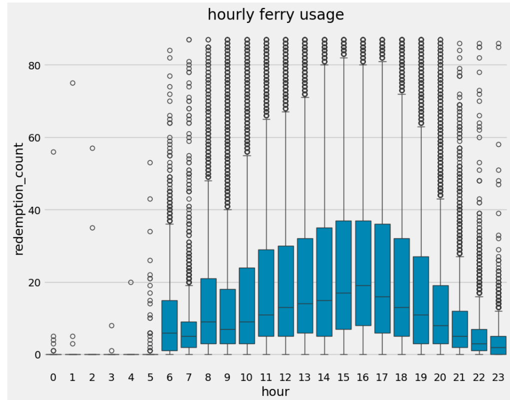
Using IQR technique, I remove the outliers and re-analyze the data.

```
# As we can see from the redemption_count column, there are some extreme outliers.
# Let's try to clean that up using IQR.

Q1 = df['redemption_count'].quantile(0.25)
Q3 = df['redemption_count'].quantile(0.75)
IQR = Q3 - Q1
lower = Q1 - 1.5*IQR
upper = Q3 + 1.5*IQR
```

```
print(f"lower: {lower}, upper: {upper}" )
```

```
lower: -48.0, upper: 88.0
```

```
# removing outlier values outside partial years 2015 and 2025

df = df.loc[df.redemption_count > lower]
df = df.loc[df.redemption_count < upper]
```



The distribution looks much better after removal of outliers using IQR.

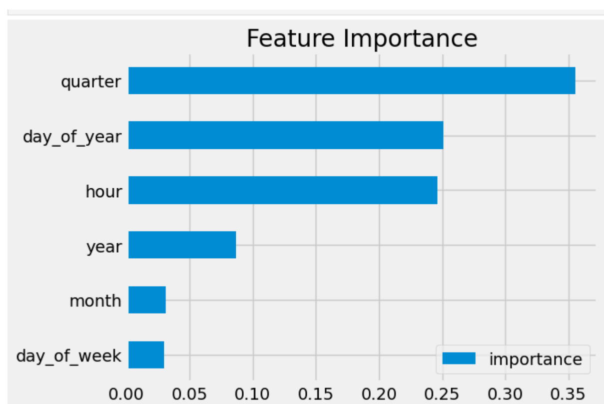
Naive Model

Next a naive model is trained using just this time series data. Below you can see the importance of each feature that was input into the model.

```
# training set is [2016 - 2023]
# test set is > 2023

train = df.loc[df.index < '01-01-2024']
test = df.loc[df.index >= '01-01-2024']

# fig, ax = plt.subplots(figsize=(15, 5))
# train.plot(ax=ax, label='Training Set', title='Data Train/Test Split')
# test.plot(ax=ax, label='Test Set')
# ax.axvline('01-01-2024', color='black', ls='--')
# ax.legend(['Training Set', 'Test Set'])
# plt.show()
```

```
FEATURES = ['day_of_year', 'hour', 'day_of_week', 'quarter', 'month', 'year']
TARGET = 'redemption_count'

X_train = train[FEATURES]
y_train = train[TARGET]

X_test = test[FEATURES]
y_test = test[TARGET]
```

```
reg = xgb.XGBRegressor(base_score=0.5, booster='gbtree',
                       n_estimators=1000,
                       early_stopping_rounds=50,
                       objective='reg:linear',
                       max_depth=3,
                       learning_rate=0.01)
reg.fit(X_train, y_train,
        eval_set=[(X_train, y_train), (X_test, y_test)],
        verbose=100)
```



Just before training of this model, the index timestamp was featureized into day_of_year, hour, day_of_week, quarter, month, year. The training set was a naive slice of any data before Jan 1, 2024. Test set was any data on or after that date.

An evaluation of this naive model shows an RMSE of 14.83. You can also see an overlay of truth and prediction data showing the predictions are far from accurate.

RMSE

```
score = np.sqrt(mean_squared_error(test['redemption_count'], test['prediction']))
print(f'RMSE Score on Test set: {score:0.2f}')
```

```
RMSE Score on Test set: 14.83
```

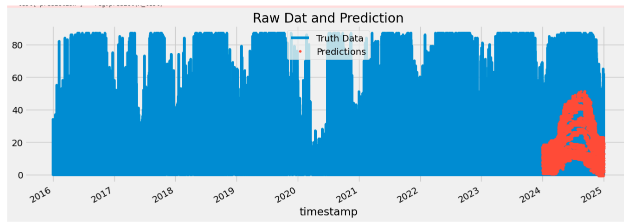
```
test['error'] = np.abs(test[TARGET] - test['prediction'])
test['date'] = test.index.date
test.groupby(['date'])['error'].mean().sort_values(ascending=False).head(10)
```

```
date
2024-07-10    21.015122
2024-05-02    20.923778
2024-03-30    19.817264
2024-07-16    19.420949
2024-03-13    19.082908
2024-04-06    18.731630
2024-04-14    18.690246
2024-05-07    18.620986
2024-06-06    18.381669
2024-06-15    18.174068
Name: error, dtype: float64
```

MAPE

```
mean_absolute_percentage_error(test['redemption_count'], test['prediction'])
```

```
18926557221355552.0
```



Additional Data

Next, I try to add some additional data to the dataset in order to improve the quality of the model. Specifically, I added weather data. While the weather dataset included a lot of columns, I only included temperature to the dataset. I also added an “is_holiday” indicator to the dataset.

```
[42]: df_weather.columns
```

```
[42]: Index(['Longitude (x)', 'Latitude (y)', 'Station Name', 'Climate ID',
       'Date/Time (LST)', 'Year', 'Month', 'Day', 'Time (LST)', 'Flag',
       'Temp (°C)', 'Temp Flag', 'Dew Point Temp (°C)', 'Dew Point Temp Flag',
       'Rel Hum (%)', 'Rel Hum Flag', 'Precip. Amount (mm)',
       'Precip. Amount Flag', 'Wind Dir (10s deg)', 'Wind Dir Flag',
       'Wind Spd (km/h)', 'Wind Spd Flag', 'Visibility (km)',
       'Visibility Flag', 'Stn Press (kPa)', 'Stn Press Flag', 'Hmdx',
       'Hmdx Flag', 'Wind Chill', 'Wind Chill Flag', 'Weather'],
      dtype='object')
```

```
[43]: df_weather = df_weather[['Year', 'Month', 'Day', 'Time (LST)', 'Temp (°C)']]
```

```
[44]: df_weather = df_weather.rename(columns={'Year': 'year',
                                              'Month': 'month',
                                              'Day': 'day_of_month',
                                              'Time (LST)': 'hour_minute',
                                              'Temp (°C)': 'temp_c'})
```

Enhanced Model

I decided to enhance the model by adding k-fold cross validation during training. This was done by using the TimeSeriesSplit method in sklearn. I decided to go with 5 folds.

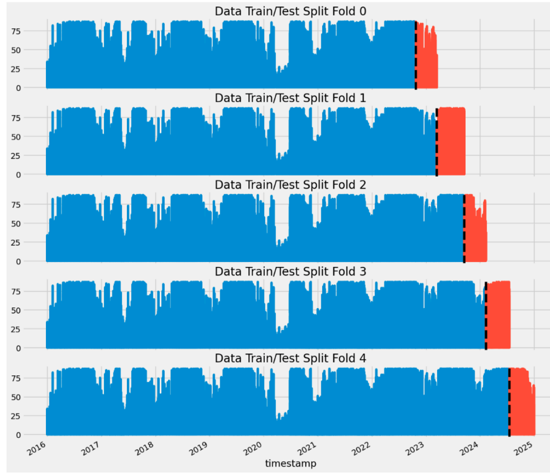
Additionally, I augment the dataset by adding lag features. Specifically, I added three lag features of the target variable with offsets of one year, two years and three years. This was done because I observed during intermediate training rounds with just 5-folds that the training results weren’t improving very much.

As per the material taught in the class, I decided to add data augmentation using lag features and went up to three lag features until the model performance started improving.

Train and Test

```
# timeseries split

tss = TimeSeriesSplit(n_splits=5, test_size=24*1*365, gap=24)
df = df.sort_index()
```

```
fig, axs = plt.subplots(5, 1, figsize=(15, 15), sharex=True)

fold = 0
for train_idx, val_idx in tss.split(df):
    train = df.iloc[train_idx]
    test = df.iloc[val_idx]
    train['redemption_count'].plot(ax=axs[fold],
                          label='Training Set',
                          title=f'Data Train/Test Split Fold {fold}')
    test['redemption_count'].plot(ax=axs[fold],
                         label='Test Set')
    axs[fold].axvline(test.index.min(), color='black', ls='--')
    fold += 1
plt.show()
```




```
def add_lags(df, target_column):
    target_map = df[target_column].to_dict()
    df['lag_1'] = (df.index - pd.Timedelta('364 days')).map(target_map)
    df['lag_2'] = (df.index - pd.Timedelta('728 days')).map(target_map)
    df['lag_3'] = (df.index - pd.Timedelta('1092 days')).map(target_map)
    return df
```

```
#### lag features

df = add_lags(df, target_column='redemption_count')
```

Model Evaluation

After performing cross-validation using 5-folds and adding three lag features, the RMSE is now 13.01. The predictions are also much closer to the the truth data. While this is better than before, this is still far from being a perfect model.

RMSE

```
score = np.sqrt(mean_squared_error(test['redemption_count'], test['prediction']))
print(f'RMSE Score on Test set: {score:0.2f}')
```

```
RMSE Score on Test set: 13.01
```

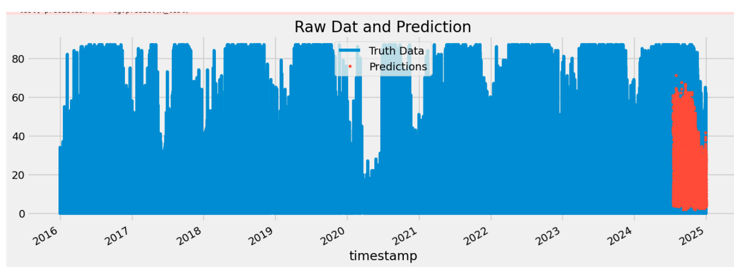
```
test['error'] = np.abs(test[TARGET] - test['prediction'])
test['date'] = test.index.date
test.groupby(['date'])['error'].mean().sort_values(ascending=False).head(10)
```

```
date
2024-09-06    21.410403
2024-09-24    18.908494
2024-09-25    16.795186
2024-09-09    16.126796
2024-10-14    15.936092
2024-10-31    14.991946
2024-10-01    14.721109
2024-08-17    14.621920
2024-12-29    14.600295
2024-08-18    14.468682
Name: error, dtype: float64
```

MAPE

```
mean_absolute_percentage_error(test['redemption_count'], test['prediction'])
```

```
1580030958239744.0
```



Conclusions

Based on the learnings from this course and the helpful guides found on kaggle, I have learned how a forecast model can be improved by continuously improving the dataset. This is the central idea of data centric machine learning. As mentioned at the start of this paper, this is the first time XGBoost has been applied to forecast transit demand for Toronto. As such, I am hoping this is considered a novel application and meets the project requirements for this course.

You will find all code related to this project at https://github.com/firozkabir/transit-forecast

References

1. Rob Mulla. (September 4, 2018). Time Series forecasting with XGBoost, version 4.
2. Toronto Island Ferry Ticket Counts. Toronto Open Data, https://open.toronto.ca/dataset/toronto-island-ferry-ticket-counts/
3. Historical Weather Data, Climate Canada, https://climate.weather.gc.ca/climate_data/daily_data_e.html
4.