Train Delay Data v2

Use this file to explore and pre-process the data

Import library's

```
import os
import pandas as pd, numpy as np, copy
import seaborn as sns
import matplotlib.pyplot as plt
from tqdm import tqdm
import tensorflow

# Set the option to display all columns, without the "..." in the middle
pd.set_option('display.max_columns', None)
```

Location of files

```
train_data_dir = 'data/delay data' # Directory where the csv data is stored
```

```
def find_csv_filenames(path_to_dir, suffix=".csv"):
    filenames = os.listdir(path_to_dir)
    return [filename for filename in filenames if filename.endswith(suffix)]

def concatenate_csv_files(directory):
    frames = []
    for subdir, dirs, files in os.walk(directory):
        for file in files:
            if file.endswith('.csv'):
                df = pd.read_csv(os.path.join(subdir, file))
                frames.append(df)
    return pd.concat(frames)

# Replace 'your_directory' with the directory you want to search
test_data = concatenate_csv_files(train_data_dir)
```

```
/var/folders/qx/bgf_wq4d7pxdmbbq4nl3gqg00000gn/T/ipykernel_990/1056983920.py:10: DtypeWarning: Columns (8) have mixed types. Specify dtype option on import or set low_memory=False
  df = pd.read_csv(os.path.join(subdir, file))
```

Encoding Values

```
columns_for_binary_encoding = []
columns_for_one_hot_encoding = []
columns_for_label_encoding = []
columns_for_target_encoding = []

labels = ['arr_at', 'pass_at', 'dep_at']

encoding_dict = {}

for column in test_data.columns:
    if test_data[column].dtype == 'object' and column not in labels:
        if len(test_data[column].unique()) == 2:
            columns_for_binary_encoding.append(column)
            encoding_dict[column] = 'Binary Encoding'

        elif len(test_data[column].unique()) > 2 and len(test_data[column].unique()) < 10:
            columns_for_one_hot_encoding.append(column)
            encoding_dict[column] = 'One Hot Encoding'

        elif len(test_data[column].unique()) > 11 and len(test_data[column].unique()) < 50:
            columns_for_label_encoding.append(column)
            encoding_dict[column] = 'Label Encoding'

        elif len(test_data[column].unique()) > 50:
            columns_for_target_encoding.append(column)
            encoding_dict[column] = 'Target Encoding'

print('Columns for Binary Encoding:', columns_for_binary_encoding)
print('Columns for One Hot Encoding:', columns_for_one_hot_encoding)
print('Columns for Label Encoding:', columns_for_label_encoding)
print('Columns for Target Encoding:', columns_for_target_encoding)
print('Columns for for y:', labels)
print('\n' + '_' * 20 + '\n')

unique_counts = pd.DataFrame.from_records(
    [(col, test_data[col].dtype, len(test_data[col].unique()), encoding_dict.get(col, 'No Encoding')) for col in test_data.columns],
    columns=['Column_Name', 'Data_Type', 'Num_Unique_Values', 'Encoding']
)

unique_counts
```

```
Columns for Binary Encoding: ['arr_atRemoved', 'pass_atRemoved', 'dep_atRemoved']
Columns for One Hot Encoding: ['dep_wet']
Columns for Label Encoding: ['tpl']
Columns for Target Encoding: ['pta', 'ptd', 'wta', 'wtp', 'wtd', 'arr_et', 'arr_wet', 'pass_et', 'dep_et']
Columns for for y: ['arr_at', 'pass_at', 'dep_at']

____________________
```

	Column_Name	Data_Type	Num_Unique_Values	Encoding
0	rid	int64	55552	No Encoding
1	tpl	object	47	Label Encoding
2	pta	object	1152	Target Encoding
3	ptd	object	1131	Target Encoding
4	wta	object	2156	Target Encoding
5	wtp	object	2297	Target Encoding
6	wtd	object	2110	Target Encoding
7	arr_et	object	1024	Target Encoding
8	arr_wet	object	786	Target Encoding
9	arr_atRemoved	object	2	Binary Encoding
10	pass_et	object	1256	Target Encoding
11	pass_wet	float64	1	No Encoding
12	pass_atRemoved	object	2	Binary Encoding
13	dep_et	object	992	Target Encoding
14	dep_wet	object	8	One Hot Encoding
15	dep_atRemoved	object	2	Binary Encoding
16	arr_at	object	1217	No Encoding
17	pass_at	object	1227	No Encoding
18	dep_at	object	1214	No Encoding
19	cr_code	float64	112	No Encoding
20	lr_code	float64	181	No Encoding

```
test_data['arr_at'].dropna().unique()
```

```
array(['07:18', '07:48', '07:57', ..., '02:20', '02:32', '02:48'],
      dtype=object)
```

Column headers

Code	Description	Notes	Importance
rid	Train RTTI Train Identifier	Unique code for train travel	

Code	Description	Notes	Importance
tpl	TIPLOC (Timing point locations)	Unique station code	
pta	Planned Time of Arrival	24hr Time value	
ptd	Planned Time of Departure	24hr Time value	
wta	Working (staff) Time of Arrival	24hr Time value- with seconds	
wtp	Working Time of Passing	24hr Time value	
wtd	Working Time of Departure	24hr Time value- with seconds	
arr_et	Estimated Arrival Time	24hr Time value	
arr_wet	Working Estimated Time	24hr Time value	
arr_atRemoved	true if actual replaced by estimated	True / False	
pass_et	Estimated Passing Time	24hr Time value	
pass_wet	Working Estimated Time	** 24hr Time value?	
arr_at	True time of arrival	24hr Time value	
pass_atRemoved	true if actual replaced by estimated	True / False	
dep_et	Estimated Departure	24hr Time value	
pass_at	True time of train passing through	24hr Time value	
dep_at	True time of train departure	24hr Time value	
dep_wet	Working Estimated Time	** 24hr Time value?	
dep_atRemoved	true if actual replaced by estimated	True / False	
arr_at	Recorded Actual Time of Arrival	24hr Time value	
pass_at	Actual Passing Time	24hr Time value	
dep_at	Actual Departure Time	24hr Time value	
cr_code	Cancellation Reason Code	Float value	
lr_code	Late Running Reason	Float Value	

Converting string time vales to timestamp

```
def convert_string_to_seconds(str):
    date_time_value = pd.to_datetime(str, format='%H:%M')
    total_seconds = date_time_value.hour * 3600 + date_time_value.minute * 60 + date_time_value.second
    return total_seconds

def convert_seconds_to_string(seconds):
    hours = seconds // 3600
    minutes = (seconds % 3600) // 60
    seconds = seconds % 60
    return "{:02d}:{:02d}:{:02d}".format(int(hours), int(minutes), int(seconds))


time= "12:34"

print(f'Orignal Value: {time}')
print(f'Converted Value: {convert_string_to_seconds(time)}')
print(f'Backwards Converted Value: {convert_seconds_to_string(convert_string_to_seconds(time))}')
```

```
Orignal Value: 12:34
Converted Value: 45240
Backwards Converted Value: 12:34:00
```

```
def convert_to_seconds(df, col, time_format):
    df[col] = pd.to_datetime(df[col], errors='coerce', format=time_format)
    seconds_since_midnight = df[col].dt.hour * 3600 + df[col].dt.minute * 60 + df[col].dt.second
    return seconds_since_midnight.fillna(-1)

# Define time columns
time_columns = test_data.columns.drop(['lr_code', 'cr_code', 'dep_atRemoved', 'pass_atRemoved', 'arr_atRemoved','tpl','rid','wta','wtd'])
time_columns_with_seconds = test_data[['wta','wtd']]

# Convert time strings to time objects for each column
for col in time_columns:
    test_data[col + '_seconds_since_midnight'] = convert_to_seconds(test_data, col, '%H:%M')
    test_data.drop(col, axis=1, inplace=True)

for col in time_columns_with_seconds:
    test_data[col + '_seconds_since_midnight'] = convert_to_seconds(test_data, col, '%H:%M:%S')
    test_data.drop(col, axis=1, inplace=True)
```

```
labels = ['arr_at_seconds_since_midnight', 'pass_at_seconds_since_midnight', 'dep_at_seconds_since_midnight']
test_data[labels]
```

	arr_at_seconds_since_midnight	pass_at_seconds_since_midnight	dep_at_seconds_since_midnight
0	-1.0	-1.0	25140.0
1	-1.0	25380.0	-1.0
2	-1.0	25440.0	-1.0
3	-1.0	25560.0	-1.0
4	-1.0	-1.0	-1.0
...	...	...	...
27013	3720.0	-1.0	3840.0
27014	-1.0	4680.0	-1.0
27015	-1.0	-1.0	-1.0
27016	-1.0	4740.0	-1.0
27017	4860.0	-1.0	-1.0

1713990 rows × 3 columns

```
import datetime

# Function to convert seconds since midnight to time value
def seconds_to_time(seconds):
    # Calculate hours, minutes, and seconds
    hours = seconds // 3600
    minutes = (seconds % 3600) // 60
    seconds = seconds % 60

    # Create a timedelta object representing the time duration
    time_delta = datetime.timedelta(hours=hours, minutes=minutes, seconds=seconds)

    # Use midnight as a reference point and add the time duration to it
    midnight = datetime.datetime.strptime('00:00:00', '%H:%M:%S').time()
    time_value = (datetime.datetime.combine(datetime.date.today(), midnight) + time_delta).time()

    return time_value

# Example usage
seconds_since_midnight = 23850.0  # Example value
time_value = seconds_to_time(seconds_since_midnight)
print("Time value:", time_value)
```

```
Time value: 06:37:30
```

Encoding the tpl

```
from sklearn.preprocessing import LabelEncoder

# create the LabelEncoder object
le = LabelEncoder()

# fit the encoder
le.fit(test_data['tpl'])

# create a DataFrame with the original and encoded values
encoding_table = pd.DataFrame({
    'Original Value': le.classes_,
    'Encoded Value': range(len(le.classes_))
})

print(encoding_table)

test_data['tpl'] = le.fit_transform(test_data['tpl'])

list_encoded_stations = test_data['tpl']

test_data.head(33)
```

	Original Value	Encoded Value
0	BOWJ	0
1	BROXBRN	1
2	BRTWOOD	2
3	BTHNLGR	3
4	CHDWLHT	4
5	CHESHNT	5
6	CHLMSFD	6
7	CLCHSTR	7
8	DISS	8
9	FRSTGT	9
10	FRSTGTJ	10
11	GIDEAPK	11
12	GIDEPKJ	12
13	GODMAYS	13
14	HAGHLYJ	14
15	HAKNYNM	15
16	HFLPEVL	16
17	HRLDWOD	17
18	ILFELEJ	18
19	ILFORD	19
20	INGTSTL	20
21	INGTSTN	21
22	IPSWEPJ	22
23	IPSWESJ	23
24	IPSWHJN	24
25	IPSWICH	25
26	KELVEDN	26
27	LIVST	27
28	MANNGTR	28
29	MANRPK	29
30	MRKSTEY	30
31	MRYLAND	31
32	NEEDHAM	32
33	NRCH	33
34	NRCHTPJ	34
35	ROMFORD	35
36	SEVNSIS	36
37	SHENFLD	37
38	STFD	38
39	STWMDGL	39
40	STWMRKT	40
41	SVNKNGS	41
42	TROWFLR	42
43	TROWSEJ	43
44	TRWSSBJ	44
45	WARE	45
46	WITHAME	46

	rid	tpl	arr_atRemoved	pass_atRemoved	dep_atRemoved	cr_code	lr_code	pta_seconds_since_midnight	ptd_seconds_since_midnight	wtp_seconds_since_midnight	arr_et_se
0	202009016712165	27	NaN	NaN	False	NaN	NaN	-1.0	25200.0	-1.0	
1	202009016712165	3	NaN	False	NaN	NaN	NaN	-1.0	-1.0	25380.0	
2	202009016712165	0	NaN	False	NaN	NaN	NaN	-1.0	-1.0	25500.0	
3	202009016712165	31	NaN	False	NaN	NaN	NaN	-1.0	-1.0	-1.0	
4	202009016712165	38	NaN	False	NaN	NaN	NaN	-1.0	-1.0	25560.0	
5	202009016712165	10	NaN	False	NaN	NaN	NaN	-1.0	-1.0	-1.0	
6	202009016712165	19	NaN	False	NaN	NaN	NaN	-1.0	-1.0	25740.0	

6	202009016712165	19	NaN	False	NaN	NaN	NaN	-1.0	-1.0	[illegible]
7	202009016712165	29	NaN	False	NaN	NaN	NaN	-1.0	-1.0	25680.0
8	202009016712165	41	NaN	False	NaN	NaN	NaN	-1.0	-1.0	-1.0
9	202009016712165	13	NaN	False	NaN	NaN	NaN	-1.0	-1.0	25800.0
10	202009016712165	4	NaN	False	NaN	NaN	NaN	-1.0	-1.0	-1.0
11	202009016712165	11	NaN	False	NaN	NaN	NaN	-1.0	-1.0	-1.0
12	202009016712165	35	NaN	False	NaN	NaN	NaN	-1.0	-1.0	25920.0
13	202009016712165	17	NaN	False	NaN	NaN	NaN	-1.0	-1.0	26100.0
14	202009016712165	2	NaN	False	NaN	NaN	NaN	-1.0	-1.0	26280.0
15	202009016712165	37	False	NaN	False	NaN	NaN	-1.0	26520.0	-1.0
16	202009016712165	6	NaN	False	NaN	NaN	NaN	-1.0	-1.0	-1.0
17	202009016712165	46	NaN	False	NaN	NaN	NaN	-1.0	-1.0	-1.0
18	202009016712165	30	NaN	False	NaN	NaN	NaN	-1.0	-1.0	-1.0
19	202009016712165	7	False	NaN	False	NaN	NaN	28200.0	28260.0	-1.0
20	202009016712165	28	False	NaN	False	NaN	NaN	28740.0	28740.0	-1.0
21	202009016712165	24	NaN	False	NaN	NaN	NaN	-1.0	-1.0	-1.0
22	202009016712165	25	False	NaN	False	NaN	NaN	29460.0	29520.0	-1.0
23	202009016712165	23	NaN	False	NaN	NaN	NaN	-1.0	-1.0	29700.0
24	202009016712165	22	NaN	False	NaN	NaN	NaN	-1.0	-1.0	29760.0
25	202009016712165	40	False	NaN	False	NaN	NaN	30180.0	30180.0	-1.0
26	202009016712165	14	NaN	False	NaN	NaN	NaN	-1.0	-1.0	-1.0
27	202009016712165	8	False	NaN	False	NaN	NaN	30900.0	30960.0	-1.0
28	202009016712165	43	NaN	False	NaN	NaN	NaN	-1.0	-1.0	31980.0
29	202009016712165	44	NaN	False	NaN	NaN	NaN	-1.0	-1.0	32040.0
30	202009016712165	34	NaN	False	NaN	NaN	NaN	-1.0	-1.0	-1.0
31	202009016712165	33	False	NaN	NaN	NaN	NaN	32160.0	-1.0	-1.0
32	202009016712168	27	NaN	NaN	False	NaN	NaN	-1.0	27000.0	-1.0

Code Values

Encoding the True / False values

True = 1

False = 0

NaN = -1

```
for col in columns_for_binary_encoding:
    # Map True to 1, False to 0, and NaN to a specific value (e.g., -1)
    test_data[col] = test_data[col].fillna(-1).astype(float)

test_data[['lr_code', 'cr_code']] = test_data[['lr_code', 'cr_code']].fillna(-1).astype(float)
```

```
test_data.sample(5)
```

	rid	tpl	arr_atRemoved	pass_atRemoved	dep_atRemoved	cr_code	lr_code	pta_seconds_since_midnight
19163	202011288006283	19	-1.0	0.0	-1.0	-1.0	-1.0	-1.0
8387	201810097681132	4	-1.0	0.0	-1.0	-1.0	-1.0	-1.0
7256	202211098750553	4	-1.0	0.0	-1.0	-1.0	-1.0	-1.0
11100	201904127628993	10	-1.0	0.0	-1.0	-1.0	-1.0	-1.0
16979	201911197671074	24	-1.0	0.0	-1.0	-1.0	-1.0	-1.0

```
unique_counts = pd.DataFrame.from_records(
    [(col, test_data[col].dtype, len(test_data[col].unique()), encoding_dict.get(col, 'No Encoding')) for col in test_data.columns],
    columns=['Column_Name', 'Data_Type', 'Num_Unique_Values', 'Encoding']
)

unique_counts
```

	Column_Name	Data_Type	Num_Unique_Values	Encoding
0	rid	int64	55552	No Encoding
1	tpl	int64	47	Label Encoding
2	arr_atRemoved	float64	2	Binary Encoding
3	pass_atRemoved	float64	2	Binary Encoding
4	dep_atRemoved	float64	2	Binary Encoding
5	cr_code	float64	112	No Encoding
6	lr_code	float64	181	No Encoding
7	pta_seconds_since_midnight	float64	1152	No Encoding
8	ptd_seconds_since_midnight	float64	1131	No Encoding
9	wtp_seconds_since_midnight	float64	1167	No Encoding
10	arr_et_seconds_since_midnight	float64	1024	No Encoding
11	arr_wet_seconds_since_midnight	float64	786	No Encoding
12	pass_et_seconds_since_midnight	float64	1256	No Encoding
13	pass_wet_seconds_since_midnight	float64	1	No Encoding
14	dep_et_seconds_since_midnight	float64	992	No Encoding
15	dep_wet_seconds_since_midnight	float64	8	No Encoding
16	arr_at_seconds_since_midnight	float64	1217	No Encoding
17	pass_at_seconds_since_midnight	float64	1227	No Encoding
18	dep_at_seconds_since_midnight	float64	1214	No Encoding
19	wta_seconds_since_midnight	float64	1074	No Encoding
20	wtd_seconds_since_midnight	float64	1080	No Encoding

Splitting the dataset

For the RNN to work it accepts data in steps. I am using the journey id 'rid' as the value of each journey $step$. Below shows there is an uneven number of step values, the minority step values will be dropped.

```
# Group by 'rid', calculate the shape of each group, and count the occurrences of each shape
shape_counts = test_data.groupby('rid').apply(lambda x: x.shape).value_counts()

# Sort the Series by the first element of the shape tuple
sorted_shape_counts = shape_counts.sort_index(key=lambda x: x.map(lambda y: y[0]))

# Print the sorted shape counts
for shape, count in sorted_shape_counts.items():
    print(f"Shape: {shape}, Count: {count}")
```

```
Shape: (1, 21), Count: 1
Shape: (2, 21), Count: 20
Shape: (3, 21), Count: 61
Shape: (4, 21), Count: 36
Shape: (5, 21), Count: 10
Shape: (6, 21), Count: 504
Shape: (7, 21), Count: 357
Shape: (8, 21), Count: 121
Shape: (9, 21), Count: 428
Shape: (10, 21), Count: 40
Shape: (11, 21), Count: 17
Shape: (12, 21), Count: 34
Shape: (13, 21), Count: 39
Shape: (14, 21), Count: 179
Shape: (15, 21), Count: 93
Shape: (16, 21), Count: 137
Shape: (17, 21), Count: 8
Shape: (18, 21), Count: 15
Shape: (19, 21), Count: 9
Shape: (20, 21), Count: 7
Shape: (21, 21), Count: 11
Shape: (22, 21), Count: 22
Shape: (23, 21), Count: 41
Shape: (24, 21), Count: 49
Shape: (25, 21), Count: 289
Shape: (26, 21), Count: 123
Shape: (27, 21), Count: 12
Shape: (28, 21), Count: 131
Shape: (29, 21), Count: 199
Shape: (30, 21), Count: 373
Shape: (31, 21), Count: 9576
Shape: (32, 21), Count: 42459
Shape: (33, 21), Count: 59
Shape: (34, 21), Count: 4
Shape: (35, 21), Count: 88
/var/folders/qx/bgf_wq4d7pxdmbbq4nl3gqg00000gn/T/ipykernel_990/800867603.py:2: DeprecationWarning: DataFrameGroupBy.apply operated on the grouping columns. This behavior is deprec
  shape_counts = test_data.groupby('rid').apply(lambda x: x.shape).value_counts()
```

```
# Get the unique 'rid' values
unique_rid = test_data['rid'].unique()

# Group by 'rid' and filter groups with shape greater than or equal to (32, 21)
filtered_test_data = test_data.groupby('rid').filter(lambda x: x.shape == (32, 21))
```

```
filtered_test_data.groupby('rid').apply(lambda x: x.shape).value_counts()
```

```
/var/folders/qx/bgf_wq4d7pxdmbbq4nl3gqg00000gn/T/ipykernel_990/4201756548.py:1: DeprecationWarning: DataFrameGroupBy.apply operated on the grouping columns. This behavior is depre
  filtered_test_data.groupby('rid').apply(lambda x: x.shape).value_counts()
(32, 21)    42459
Name: count, dtype: int64
```

```
from sklearn.preprocessing import MinMaxScaler

# Define the proportion of data to allocate to the validation set (e.g., 20%)
validation_proportion = 0.2

# Identify unique journeys based on the 'rid' column
unique_journeys = filtered_test_data['rid'].unique()

# Calculate the number of unique journeys to allocate to the validation set
num_validation_journeys = int(len(unique_journeys) * validation_proportion)

# Select a subset of unique journeys for validation
validation_journeys = unique_journeys[-num_validation_journeys:]

# Split the data into train and validation sets based on the selected unique journeys
train_df = filtered_test_data[~filtered_test_data['rid'].isin(validation_journeys)]
validation_df = filtered_test_data[filtered_test_data['rid'].isin(validation_journeys)]

# Drop the 'rid' column from both dataframes
train_df = train_df.drop(columns=['rid'])
validation_df = validation_df.drop(columns=['rid'])

labels = ['arr_at_seconds_since_midnight', 'pass_at_seconds_since_midnight', 'dep_at_seconds_since_midnight']

# Split the train data into X and y
X_train = train_df.drop(columns=[labels[0]])
y_train = train_df[labels[0]]

_X_val = validation_df

# Split the validation data into X and y
X_val = validation_df.drop(columns=[labels[0]])
y_val = validation_df[labels[0]]


#-------- Scaling the values negatively affects the model.. --------

# features_to_scale = time_columns.to_list() + time_columns_with_seconds.columns.to_list()
# features_to_scale.remove('arr_at')
# features_to_scale = [column + '_seconds_since_midnight' for column in features_to_scale]

# # Create the scaler
# scaler = MinMaxScaler(feature_range=(0, 1))

# # Replace -1 with NaN
# X_train.replace(-1, np.nan, inplace=True)
# X_val.replace(-1, np.nan, inplace=True)

# # Fit on the training data
# scaler.fit(X_train[features_to_scale])

# # Transform both the training and validation data
# X_train[features_to_scale] = scaler.transform(X_train[features_to_scale])
# X_train.fillna(-1, inplace=True)  # Replace NaNs with -1

# X_val[features_to_scale] = scaler.transform(X_val[features_to_scale])
```

```
# X_val.fillna(-1, inplace=True)  # Replace NaNs with -1

X_val
```

	tpl	arr_atRemoved	pass_atRemoved	dep_atRemoved	cr_code	lr_code	pta_seconds_since_midnight	ptd_seconds_sir
9110	27	-1.0	-1.0	0.0	-1.0	-1.0	-1.0	
9111	3	-1.0	0.0	-1.0	-1.0	-1.0	-1.0	
9112	0	-1.0	0.0	-1.0	-1.0	-1.0	-1.0	
9113	38	-1.0	0.0	-1.0	-1.0	-1.0	-1.0	
9114	31	-1.0	0.0	-1.0	-1.0	-1.0	-1.0	
...	...	...	...	...	...	...	...	
27013	8	0.0	-1.0	0.0	-1.0	-1.0	3780.0	
27014	43	-1.0	0.0	-1.0	-1.0	-1.0	-1.0	
27015	44	-1.0	0.0	-1.0	-1.0	-1.0	-1.0	
27016	34	-1.0	0.0	-1.0	-1.0	-1.0	-1.0	
27017	33	0.0	-1.0	-1.0	-1.0	-1.0	4920.0	

271712 rows × 19 columns

Developing The Model

RNN

```
num_stations = len(train_df['lr_code'].unique())
num_features = train_df.drop(columns=['arr_at_seconds_since_midnight']).shape[1]
num_samples = train_df.shape[0]

print("Shape of array before reshaping:", train_df.drop(columns=['arr_et_seconds_since_midnight']).values.shape)
print("num_stations:", num_stations)
print("num_features:", num_features)
print("num_samples:", num_samples)


num_val_stations = len(validation_df['lr_code'].unique())
num_val_features = validation_df.drop(columns=['arr_at_seconds_since_midnight']).shape[1]
num_val_samples = validation_df.shape[0]

print("\n\nShape of array before reshaping:", validation_df.drop(columns=['arr_et_seconds_since_midnight']).values.shape)
print("num_stations:", num_val_stations)
print("num_features:", num_val_features)
print("num_samples:", num_val_samples)
```

```
Shape of array before reshaping: (1086976, 19)
num_stations: 159
num_features: 19
num_samples: 1086976


Shape of array before reshaping: (271712, 19)
num_stations: 108
num_features: 19
num_samples: 271712
```

```
# Reshape the data
X_train_3d = X_train.values.reshape((-1, 32, X_train.shape[1]))
y_train_3d = y_train.values.reshape((-1, 32, 1))

X_val_3d = X_val.values.reshape((-1, 32, X_val.shape[1]))
y_val_3d = y_val.values.reshape((-1, 32, 1))
```

```
from keras.models import Sequential
from keras.layers import LSTM, TimeDistributed, Dense
from keras.callbacks import EarlyStopping, TensorBoard
from keras.metrics import MeanSquaredLogarithmicError, MeanAbsolutePercentageError
import tensorflow as tf
import datetime

# Define RMSE
def rmse(y_true, y_pred):
    return tf.sqrt(tf.reduce_mean(tf.square(y_pred - y_true)))

# Define the LSTM model
model = Sequential()
model.add(LSTM(50, activation='relu', return_sequences=True, input_shape=(32, X_train.shape[1])))
model.add(TimeDistributed(Dense(1)))
model.compile(optimizer='adam', loss='mse', metrics=['mae', rmse, MeanSquaredLogarithmicError(), MeanAbsolutePercentageError()])

# Define early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=5)

# Define TensorBoard
time_stamp = datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
log_dir = "logs/fit/" + time_stamp
tensorboard_callback = TensorBoard(log_dir=log_dir, histogram_freq=1)

model.summary()

# Train the model
model.fit(X_train_3d, y_train_3d, epochs=100, verbose=1, validation_data=(X_val_3d, y_val_3d), callbacks=[early_stopping, tensorboard_callback])

model.save(f'RNN Model_{time_stamp}.keras')
```

```
2024-04-25 09:38:05.094060: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compile
/Users/joshuanewton/Library/CloudStorage/OneDrive-UniversityofEastAnglia/Modules/Artificial Intelligence/Assignments/
  super().__init__(**kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 32, 50)	14,000
time_distributed (TimeDistributed)	(None, 32, 1)	51

Total params: 14,051 (54.89 KB)
Trainable params: 14,051 (54.89 KB)
Non-trainable params: 0 (0.00 B)

```
Epoch 1/100
1062/1062 ━━━━━━━━━━━━━━━━━━━━ 23s 17ms/step - loss: 281791648.0000 - mae: 9478.1768 - mean_absolute_percentage_erro
Epoch 2/100
1062/1062 ━━━━━━━━━━━━━━━━━━━━ 16s 15ms/step - loss: 29075462.0000 - mae: 3335.2114 - mean_absolute_percentage_error:
Epoch 3/100
1062/1062 ━━━━━━━━━━━━━━━━━━━━ 15s 14ms/step - loss: 21914710.0000 - mae: 2742.9531 - mean_absolute_percentage_error:
Epoch 4/100
1062/1062 ━━━━━━━━━━━━━━━━━━━━ 15s 14ms/step - loss: 18236650.0000 - mae: 2468.5701 - mean_absolute_percentage_error:
Epoch 5/100
1062/1062 ━━━━━━━━━━━━━━━━━━━━ 16s 15ms/step - loss: 14661331.0000 - mae: 2311.2205 - mean_absolute_percentage_error:
Epoch 6/100
1062/1062 ━━━━━━━━━━━━━━━━━━━━ 15s 14ms/step - loss: 10282062.0000 - mae: 1931.9944 - mean_absolute_percentage_error:
Epoch 7/100
1062/1062 ━━━━━━━━━━━━━━━━━━━━ 17s 16ms/step - loss: 7713258.0000 - mae: 1519.1952 - mean_absolute_percentage_error:
Epoch 8/100
1062/1062 ━━━━━━━━━━━━━━━━━━━━ 17s 16ms/step - loss: 6578804.5000 - mae: 1220.2916 - mean_absolute_percentage_error:
Epoch 9/100
1062/1062 ━━━━━━━━━━━━━━━━━━━━ 17s 16ms/step - loss: 5355782.0000 - mae: 984.2577 - mean_absolute_percentage_error:
Epoch 10/100
1062/1062 ━━━━━━━━━━━━━━━━━━━━ 15s 14ms/step - loss: 4654697.0000 - mae: 779.4251 - mean_absolute_percentage_error:
Epoch 11/100
1062/1062 ━━━━━━━━━━━━━━━━━━━━ 19s 17ms/step - loss: 4767369.5000 - mae: 751.3188 - mean_absolute_percentage_error:
Epoch 12/100
1062/1062 ━━━━━━━━━━━━━━━━━━━━ 19s 18ms/step - loss: 3661168.7500 - mae: 561.9827 - mean_absolute_percentage_error:
Epoch 13/100
1062/1062 ━━━━━━━━━━━━━━━━━━━━ 16s 15ms/step - loss: 3083715.2500 - mae: 445.0934 - mean_absolute_percentage_error:
Epoch 14/100
1062/1062 ━━━━━━━━━━━━━━━━━━━━ 15s 15ms/step - loss: 3273831.2500 - mae: 485.1795 - mean_absolute_percentage_error:
Epoch 15/100
1062/1062 ━━━━━━━━━━━━━━━━━━━━ 17s 16ms/step - loss: 2853895.7500 - mae: 336.4337 - mean_absolute_percentage_error:
Epoch 16/100
1062/1062 ━━━━━━━━━━━━━━━━━━━━ 15s 14ms/step - loss: 2541072.0000 - mae: 307.1745 - mean_absolute_percentage_error:
Epoch 17/100
1062/1062 ━━━━━━━━━━━━━━━━━━━━ 14s 13ms/step - loss: 2253947.2500 - mae: 296.9044 - mean_absolute_percentage_error:
Epoch 18/100
1062/1062 ━━━━━━━━━━━━━━━━━━━━ 15s 14ms/step - loss: 2644528.7500 - mae: 321.8714 - mean_absolute_percentage_error:
Epoch 19/100
1062/1062 ━━━━━━━━━━━━━━━━━━━━ 15s 14ms/step - loss: 2289504.7500 - mae: 300.9827 - mean_absolute_percentage_error:
Epoch 20/100
1062/1062 ━━━━━━━━━━━━━━━━━━━━ 21s 19ms/step - loss: 2188565.7500 - mae: 306.9738 - mean_absolute_percentage_error:
Epoch 21/100
1062/1062 ━━━━━━━━━━━━━━━━━━━━ 17s 16ms/step - loss: 1745455.2500 - mae: 219.4247 - mean_absolute_percentage_error:
```

```
raise SystemExit("Stop right there!")
```

```
An exception has occurred, use %tb to see the full traceback.

SystemExit: Stop right there!

/Users/joshuanewton/Library/CloudStorage/OneDrive-UniversityofEastAnglia/Modules/Artificial Intelligence/Assignments/
  warn("To exit: use 'exit', 'quit', or Ctrl-D.", stacklevel=1)
```

```
# Use the below to load the model instead of training again..

# from tensorflow.keras.models import load_model
# from tensorflow.keras import backend as K

# # Define the custom RMSE function
# def rmse(y_true, y_pred):
#     return K.sqrt(K.mean(K.square(y_pred - y_true)))

# # Load the model
# model = load_model('RNN Model.keras', custom_objects={'rmse': rmse})


from sklearn.metrics import mean_squared_error
from math import sqrt

# Make predictions
y_pred = model.predict(X_val_3d)

# Replace values between -100 and 100 with -1
y_pred = np.where((y_pred > -100) & (y_pred < 100), -1, y_pred)

_X_val['my_prediction_since_midnight'] = y_pred.flatten()

_X_val['arr_at'] = _X_val['arr_at_seconds_since_midnight'].apply(convert_seconds_to_string)
_X_val['my_prediction'] = _X_val['my_prediction_since_midnight'].apply(convert_seconds_to_string)

_X_val_filtered = _X_val[_X_val['arr_at_seconds_since_midnight'] != -1].copy()

# Calculate the score of the predictions
mse_score = mean_squared_error(_X_val_filtered['arr_at_seconds_since_midnight'], _X_val_filtered['my_prediction_since_midnight'])

# Calculate RMSE
rmse_score = sqrt(mse_score)

print(f"Prediction MSE score: {mse_score}")
print(f"Prediction RMSE score: {rmse_score}")
print(f"Prediction RMSE score in seconds: {convert_seconds_to_string(rmse_score)}")

# Calculate the difference in time between the 'arr_at' and 'my_prediction' columns
_X_val_filtered.loc[:, 'time_difference'] = abs(_X_val_filtered['arr_at_seconds_since_midnight'] - _X_val_filtered['my_prediction_since_midnight'])

# Convert 'time_difference' to HH:MM:SS format
_X_val_filtered.loc[:, 'time_difference'] = _X_val_filtered['time_difference'].apply(convert_seconds_to_string)

_X_val_filtered[['arr_at_seconds_since_midnight', 'my_prediction_since_midnight', 'arr_at', 'my_prediction', 'time_difference']]
```

```
266/266 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step
Prediction MSE score: 5542662.705715629
Prediction RMSE score: 2354.2860288664224
Prediction RMSE score in seconds: 00:39:14
/var/folders/qx/bgf_wq4d7pxdmbbq4nl3gqg00000gn/T/ipykernel_990/955898833.py:31: FutureWarning: Setting an item of in
  _X_val_filtered.loc[:, 'time_difference'] = _X_val_filtered['time_difference'].apply(convert_seconds_to_string)
```

	arr_at_seconds_since_midnight	my_prediction_since_midnight	arr_at	my_prediction	time_difference
9129	67440.0	67510.789062	18:44:00	18:45:10	00:01:10
9130	68040.0	68186.421875	18:54:00	18:56:26	00:02:26
9132	68760.0	68894.117188	19:06:00	19:08:14	00:02:14
9137	70140.0	69934.726562	19:29:00	19:25:34	00:03:25
9141	71340.0	71757.664062	19:49:00	19:55:57	00:06:57
...	...	...	...	...	...
27006	1620.0	706.702148	00:27:00	00:11:46	00:15:13
27008	2220.0	2573.316406	00:37:00	00:42:53	00:05:53
27011	3000.0	3078.495117	00:50:00	00:51:18	00:01:18
27013	3720.0	3781.103027	01:02:00	01:03:01	00:01:01
27017	4860.0	4895.019043	01:21:00	01:21:35	00:00:35

58492 rows × 5 columns

MLP

```
X_train
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[1], line 1
----> 1 X_train

NameError: name 'X_train' is not defined
```

```
from keras.models import Sequential
from keras.layers import Dense
from keras.callbacks import EarlyStopping

# Define the model
model = Sequential()
model.add(Dense(32, input_dim=X_train.shape[1], activation='relu'))
model.add(Dense(16, activation='relu'))
model.add(Dense(1, activation='linear'))

# Compile the model
model.compile(loss='mean_squared_error', optimizer='adam', metrics=['mean_absolute_error', 'mean_squared_error'])
```

```
2024-04-24 10:11:32.183118: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in performance-critical oper
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
/Users/joshuanewton/Library/CloudStorage/OneDrive-UniversityofEastAnglia/Modules/Artificial Intelligence/Assignments/Assignment 02/Chat_bot/.venv/lib/python3.11/site-packages/kera
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
# Define the early stopping criteria
early_stopping = EarlyStopping(monitor='val_loss', patience=5)

# Fit the model (assuming you have training and validation data defined)
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=100, callbacks=[early_stopping])

model.save('MLP Model.keras')

# Evaluate the model
loss = model.evaluate(X_test, y_test)
print('Test loss:', loss)
```

```
Epoch 1/100
690/690 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 106773280.0000 - mean_absolute_error: 3880.6829 - mean_squared_error: 106773280.0000 - val_loss: 8124206.0000 - val_mean_absolute_
Epoch 2/100
690/690 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 8974206.0000 - mean_absolute_error: 513.1395 - mean_squared_error: 8974206.0000 - val_loss: 4720593.0000 - val_mean_absolute_erro
Epoch 3/100
690/690 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 5071497.0000 - mean_absolute_error: 340.0338 - mean_squared_error: 5071497.0000 - val_loss: 4869332.0000 - val_mean_absolute_erro
Epoch 4/100
690/690 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 6484090.0000 - mean_absolute_error: 336.0281 - mean_squared_error: 6484090.0000 - val_loss: 4163950.5000 - val_mean_absolute_erro
Epoch 5/100
690/690 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 6462429.0000 - mean_absolute_error: 327.1425 - mean_squared_error: 6462429.0000 - val_loss: 4283006.5000 - val_mean_absolute_erro
Epoch 6/100
690/690 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 4189777.2500 - mean_absolute_error: 276.3677 - mean_squared_error: 4189777.2500 - val_loss: 3847919.7500 - val_mean_absolute_erro
Epoch 7/100
690/690 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 5504645.5000 - mean_absolute_error: 276.4265 - mean_squared_error: 5504645.5000 - val_loss: 3808758.0000 - val_mean_absolute_erro
Epoch 8/100
690/690 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 5703067.0000 - mean_absolute_error: 345.7055 - mean_squared_error: 5703067.0000 - val_loss: 3802681.0000 - val_mean_absolute_erro
Epoch 9/100
690/690 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 6623099.5000 - mean_absolute_error: 326.4132 - mean_squared_error: 6623099.5000 - val_loss: 4007859.5000 - val_mean_absolute_erro
Epoch 10/100
690/690 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 3609464.5000 - mean_absolute_error: 270.9659 - mean_squared_error: 3609464.5000 - val_loss: 3743166.5000 - val_mean_absolute_erro
Epoch 11/100
690/690 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 5642946.0000 - mean_absolute_error: 329.8374 - mean_squared_error: 5642946.0000 - val_loss: 3722035.2500 - val_mean_absolute_erro
Epoch 12/100
690/690 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 3827563.7500 - mean_absolute_error: 281.3314 - mean_squared_error: 3827563.7500 - val_loss: 3721890.2500 - val_mean_absolute_erro
Epoch 13/100
690/690 ━━━━━━━━━━━━━━━━━━━━ 1s 957us/step - loss: 4076675.2500 - mean_absolute_error: 246.8627 - mean_squared_error: 4076675.2500 - val_loss: 4222328.5000 - val_mean_absolute_er
Epoch 14/100
690/690 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 3766420.2500 - mean_absolute_error: 322.1334 - mean_squared_error: 3766420.2500 - val_loss: 3694875.7500 - val_mean_absolute_erro
Epoch 15/100
690/690 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 5517638.5000 - mean_absolute_error: 309.6285 - mean_squared_error: 5517638.5000 - val_loss: 3538413.7500 - val_mean_absolute_erro
Epoch 16/100
690/690 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 2727532.5000 - mean_absolute_error: 235.2153 - mean_squared_error: 2727532.5000 - val_loss: 3586469.7500 - val_mean_absolute_erro
Epoch 17/100
690/690 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 3413464.5000 - mean_absolute_error: 231.2191 - mean_squared_error: 3413464.5000 - val_loss: 3806154.0000 - val_mean_absolute_erro
Epoch 18/100
690/690 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 5648635.5000 - mean_absolute_error: 276.9623 - mean_squared_error: 5648635.5000 - val_loss: 3981994.2500 - val_mean_absolute_erro
Epoch 19/100
690/690 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 4799237.0000 - mean_absolute_error: 322.9114 - mean_squared_error: 4799237.0000 - val_loss: 3848264.5000 - val_mean_absolute_erro
Epoch 20/100
690/690 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 4739511.0000 - mean_absolute_error: 302.5638 - mean_squared_error: 4739511.0000 - val_loss: 3431392.2500 - val_mean_absolute_erro
Epoch 21/100
690/690 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 4818973.5000 - mean_absolute_error: 293.7706 - mean_squared_error: 4818973.5000 - val_loss: 3624541.5000 - val_mean_absolute_erro
```

```
Epoch 22/100
690/690 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 3617766.5000 - mean_absolute_error: 276.5949 - mean_squared_error: 3617766.5000 - val_loss: 4049440.7500 - val_mean_absolute_erro
Epoch 23/100
690/690 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 4634426.5000 - mean_absolute_error: 301.6610 - mean_squared_error: 4634426.5000 - val_loss: 3374576.2500 - val_mean_absolute_erro
Epoch 24/100
690/690 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 3399142.0000 - mean_absolute_error: 221.6506 - mean_squared_error: 3399142.0000 - val_loss: 3716261.5000 - val_mean_absolute_erro
Epoch 25/100
690/690 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 3663961.0000 - mean_absolute_error: 266.8208 - mean_squared_error: 3663961.0000 - val_loss: 3239872.7500 - val_mean_absolute_erro
Epoch 26/100
690/690 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 4369014.5000 - mean_absolute_error: 265.0534 - mean_squared_error: 4369014.5000 - val_loss: 3780497.7500 - val_mean_absolute_erro
Epoch 27/100
690/690 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 4395955.0000 - mean_absolute_error: 294.4192 - mean_squared_error: 4395955.0000 - val_loss: 3612268.2500 - val_mean_absolute_erro
Epoch 28/100
690/690 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - loss: 5014749.0000 - mean_absolute_error: 305.7670 - mean_squared_error: 5014749.0000 - val_loss: 3330054.0000 - val_mean_absolute_erro
Epoch 29/100
690/690 ━━━━━━━━━━━━━━━━━━━━ 1s 950us/step - loss: 2628512.0000 - mean_absolute_error: 208.6415 - mean_squared_error: 2628512.0000 - val_loss: 3213122.2500 - val_mean_absolute_er
```

```
# preds = model.predict(X_test)


# Assuming y_test is a pandas Series or DataFrame
df = pd.DataFrame(y_test)

# Add a new column with the model predictions
df['predictions'] = model.predict(X_test)
df['real_time'] = df['arr_at_seconds_since_midnight'].apply(seconds_to_time)
df['predictions_time'] = df['predictions'].apply(seconds_to_time)
```

```
173/173 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
```

```
df
```

	arr_at_seconds_since_midnight	predictions	real_time	predictions_time
8238	-1.0	1.044594	23:59:59	00:00:01.044594
3690	-1.0	17.069893	23:59:59	00:00:17.069893
6528	-1.0	14.059761	23:59:59	00:00:14.059761
9579	31200.0	31250.390625	08:40:00	08:40:50.390625
7786	-1.0	20.427315	23:59:59	00:00:20.427315
...	...	...	...	...
10829	2700.0	2840.061523	00:45:00	00:47:20.061523
14105	-1.0	115.358467	23:59:59	00:01:55.358467
5130	-1.0	6.540596	23:59:59	00:00:06.540596
15974	-1.0	18.718025	23:59:59	00:00:18.718025
19050	-1.0	17.253487	23:59:59	00:00:17.253487

5515 rows × 4 columns

Creating dummy DF with a fake scenario

```
example__single_journey = filtered_test_data[filtered_test_data['rid'] == unique_rid[25]]
example__single_journey = example__single_journey.reset_index(drop=True)

example__single_journey
```

	rid	tpl	arr_atRemoved	pass_atRemoved	dep_atRemoved	cr_code	lr_code	pta_seconds_since_midnight	pt
0	202009018006357	27	-1.0	-1.0	0.0	-1.0	-1.0	-1.0	
1	202009018006357	3	-1.0	0.0	-1.0	-1.0	-1.0	-1.0	
2	202009018006357	0	-1.0	0.0	-1.0	-1.0	-1.0	-1.0	
3	202009018006357	38	0.0	-1.0	0.0	-1.0	-1.0	-1.0	
4	202009018006357	31	-1.0	0.0	-1.0	-1.0	-1.0	-1.0	
5	202009018006357	10	-1.0	0.0	-1.0	-1.0	-1.0	-1.0	
6	202009018006357	19	-1.0	0.0	-1.0	-1.0	-1.0	-1.0	
7	202009018006357	29	-1.0	0.0	-1.0	-1.0	-1.0	-1.0	
8	202009018006357	41	-1.0	0.0	-1.0	-1.0	-1.0	-1.0	
9	202009018006357	13	-1.0	0.0	-1.0	-1.0	-1.0	-1.0	
10	202009018006357	4	-1.0	0.0	-1.0	-1.0	-1.0	-1.0	
11	202009018006357	11	-1.0	0.0	-1.0	-1.0	-1.0	-1.0	
12	202009018006357	35	-1.0	0.0	-1.0	-1.0	-1.0	-1.0	
13	202009018006357	17	-1.0	0.0	-1.0	-1.0	-1.0	-1.0	
14	202009018006357	2	-1.0	0.0	-1.0	-1.0	-1.0	-1.0	
15	202009018006357	37	-1.0	0.0	-1.0	-1.0	-1.0	-1.0	
16	202009018006357	6	0.0	-1.0	0.0	-1.0	-1.0	79500.0	
17	202009018006357	46	-1.0	0.0	-1.0	-1.0	-1.0	-1.0	
18	202009018006357	30	-1.0	0.0	-1.0	-1.0	-1.0	-1.0	
19	202009018006357	7	0.0	-1.0	0.0	-1.0	-1.0	80700.0	
20	202009018006357	28	0.0	-1.0	0.0	-1.0	-1.0	81180.0	
21	202009018006357	24	-1.0	0.0	-1.0	-1.0	-1.0	-1.0	
22	202009018006357	25	0.0	-1.0	0.0	-1.0	-1.0	81960.0	
23	202009018006357	23	-1.0	0.0	-1.0	-1.0	-1.0	-1.0	
24	202009018006357	22	-1.0	0.0	-1.0	-1.0	-1.0	-1.0	
25	202009018006357	40	0.0	-1.0	0.0	-1.0	-1.0	82680.0	
26	202009018006357	14	-1.0	0.0	-1.0	-1.0	-1.0	-1.0	
27	202009018006357	8	0.0	-1.0	0.0	-1.0	-1.0	83460.0	
28	202009018006357	43	-1.0	0.0	-1.0	-1.0	-1.0	-1.0	
29	202009018006357	44	-1.0	0.0	-1.0	-1.0	-1.0	-1.0	
30	202009018006357	34	-1.0	0.0	-1.0	-1.0	-1.0	-1.0	

```
print(
    convert_seconds_to_string(
        (example__single_journey.loc[22, 'ptd_seconds_since_midnight']
        -
        example__single_journey.loc[0, 'ptd_seconds_since_midnight'])

        - (convert_string_to_seconds('12:00'))

        )
)
```

```
    -11:17:00
```

```
"""
Cell generated by Data Wrangler.
"""
def clean_data(example__single_journey):
    # Drop column: 'rid'
    example__single_journey = example__single_journey.drop(columns=['rid'])
    cols_to_blank = example__single_journey.columns
    cols_to_blank = cols_to_blank[1:]
    example__single_journey[cols_to_blank] = -1
    return example__single_journey

example__single_journey_clean = clean_data(example__single_journey.copy())
example__single_journey_clean = example__single_journey_clean.reset_index(drop=True)
# example__single_journey_clean
```

```
example__single_journey_clean.at[22, 'arr_atRemoved'] = 0
example__single_journey_clean.at[22, 'pass_atRemoved'] = 0
example__single_journey_clean.at[22, 'dep_atRemoved'] = 0
example__single_journey_clean.at[22, 'pta_seconds_since_midnight'] = convert_string_to_seconds('12:00')
example__single_journey_clean.at[22, 'ptd_seconds_since_midnight'] = convert_string_to_seconds('12:09')
example__single_journey_clean.at[22, 'arr_et_seconds_since_midnight'] = convert_string_to_seconds('12:00')
example__single_journey_clean.at[22, 'dep_et_seconds_since_midnight'] = convert_string_to_seconds('12:06')
example__single_journey_clean.at[22, 'arr_at_seconds_since_midnight'] = convert_string_to_seconds('12:01')
example__single_journey_clean.at[22, 'dep_at_seconds_since_midnight'] = convert_string_to_seconds('12:15')

example__single_journey_clean.at[0, 'arr_atRemoved'] = 0
example__single_journey_clean.at[0, 'pass_atRemoved'] = 0
example__single_journey_clean.at[0, 'dep_atRemoved'] = 0
example__single_journey_clean.at[0, 'pta_seconds_since_midnight'] = convert_string_to_seconds('11:17')
example__single_journey_clean.at[0, 'ptd_seconds_since_midnight'] = convert_string_to_seconds('11:20')
example__single_journey_clean.at[0, 'arr_et_seconds_since_midnight'] = convert_string_to_seconds('11:17')
example__single_journey_clean.at[0, 'dep_et_seconds_since_midnight'] = convert_string_to_seconds('11:19')
example__single_journey_clean.at[0, 'arr_at_seconds_since_midnight'] = convert_string_to_seconds('11:18')
example__single_journey_clean.at[0, 'dep_at_seconds_since_midnight'] = convert_string_to_seconds('11:27')
```

```
example__single_journey_clean_1 = clean_data(example__single_journey_clean.copy())
example__single_journey_clean_1 = example__single_journey_clean_1.drop(columns=['arr_at_seconds_since_midnight'], axis=1)

example__single_journey_clean_1
```