ProcessingV2

April 28, 2024

[26]:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import scipy
import sklearn
import os
```

[27]:

```
def find_csv_files(folder_path, suffix=".csv"):
    """
    Find all CSV files in the given folder path.

    Args:
    - folder_path (str): Path to the folder where CSV files are located.
    - suffix (str): Suffix to filter files (default is '.csv').

    Returns:
    - list: List of CSV files with the given suffix.
    """
    csv_files = []
    for root, dirs, files in os.walk(folder_path):
        for file in files:
            if file.endswith(suffix):
                csv_files.append(os.path.join(root, file))
    return csv_files
```

[28]:

```
def plot_top_n_categorical(data, column, n=None, figsize=(15, 5)):
    """
    Plot the distribution of the top N categories of a categorical variable.

    Args:
    - data (pandas DataFrame): DataFrame containing the categorical variable.
    - column (str): Name of the categorical variable column.
    - n (int or None): Number of top categories to plot. If None, plot all␣
 ↪categories (default is None).
    - figsize (tuple): Width and height of the figure in inches (default is␣
 ↪(10, 6)).
```

```
    Returns:
    - None (displays the plot).
    """
    if n is None:
        categories = data[column].value_counts().index
    else:
        categories = data[column].value_counts().nlargest(n).index

    data_filtered = data[data[column].isin(categories)]

    plt.figure(figsize=figsize)
    sns.countplot(data=data_filtered, x=column, order=categories)
    plt.title(f'Top {len(categories)} Categories of {column}')
    plt.xlabel(column)
    plt.ylabel('Count')
    plt.xticks(rotation=90)
    plt.show()
```

[29]:

```
data = pd.read_csv("./INITIAL_PROCESSED_DATA.csv")
```

[30]:

```
data = data.sort_values(by=['Date (MM/DD/YYYY)', 'Scheduled Arrival Time'])
```

[31]:

```
to_drop = [
    'Arrival Delay (Minutes)',
    'Flight Number',
    'Tail Number',
]
```

[32]:

```
data = data.drop(columns=to_drop)
```

[33]:

```
weather_data_path = './imputed_weather/{}_weather_data.csv'
```

[34]:

```
data.head()
```

[34]:

```
  Carrier Code Date (MM/DD/YYYY) Origin Airport Scheduled Arrival Time  \
0           B6        2010-01-01            JFK                  00:01
1           B6        2010-01-01            JFK                  08:55
2           MQ        2010-01-01            ORD                  11:20
3           9E        2010-01-01            DTW                  11:44
4           B6        2010-01-01            JFK                  11:52

   Scheduled Elapsed Time (Minutes) FLIGHT_STATUS  month  day  season WeekDay
0                                76          LATE      1    1  winter  Friday
1                                75          LATE      1    1  winter  Friday
2                               100        ONTIME      1    1  winter  Friday
3                                84          LATE      1    1  winter  Friday
```

```
4                                71          LATE      1    1  winter  Friday
```

```
[35]: data['Scheduled Arrival Time'] = data['Scheduled Arrival Time'].replace('24:
      ↪00', '23:59')
```

```
[36]: data['UNIX_DATE'] = pd.to_datetime(data['Date (MM/DD/YYYY)'] + ' ' +␣
      ↪data['Scheduled Arrival Time'])
```

```
[37]: data['UNIX_TIMESTAMP'] = data['UNIX_DATE'].apply(lambda x: int(x.timestamp()))
```

```
[38]: data.head()
```

```
[38]:   Carrier Code Date (MM/DD/YYYY) Origin Airport Scheduled Arrival Time  \
      0           B6        2010-01-01            JFK                  00:01
      1           B6        2010-01-01            JFK                  08:55
      2           MQ        2010-01-01            ORD                  11:20
      3           9E        2010-01-01            DTW                  11:44
      4           B6        2010-01-01            JFK                  11:52

         Scheduled Elapsed Time (Minutes) FLIGHT_STATUS  month  day  season WeekDay  \
      0                                76          LATE      1    1  winter  Friday
      1                                75          LATE      1    1  winter  Friday
      2                               100        ONTIME      1    1  winter  Friday
      3                                84          LATE      1    1  winter  Friday
      4                                71          LATE      1    1  winter  Friday

                  UNIX_DATE  UNIX_TIMESTAMP
      0 2010-01-01 00:01:00      1262304060
      1 2010-01-01 08:55:00      1262336100
      2 2010-01-01 11:20:00      1262344800
      3 2010-01-01 11:44:00      1262346240
      4 2010-01-01 11:52:00      1262346720
```

1 Weather

```
[39]: def mode_imputer(data):
          """
          Perform mode imputation on a DataFrame.

          Parameters:
              data (DataFrame): Input DataFrame with missing values.

          Returns:
              DataFrame: DataFrame with missing values replaced by mode.
          """
          # Fill missing values with mode
```

```
    data_imputed = data.fillna(data.mode().iloc[0])

    return data_imputed
```

```
[40]: all_weather_data = find_csv_files('./imputed_weather/')
```

```
[ ]:
```

```
[41]: # for d in all_weather_data:
      #     dd = mode_imputer(pd.read_csv(d))
      #     print(f"============{d}===============")
      #     dd = dd.fillna(1013)
      #     print(dd.isna().sum())
      #     print("===========================")
```

```
[42]: weather1 = pd.read_csv(all_weather_data[0], parse_dates=['date'])
```

```
[43]: weather1['UNIX_TIMESTAMP'] = weather1['date'].apply(lambda x: int(x.
        ↪timestamp()))
```

```
[44]: weather1.head()
```

```
[44]:   station                date  latitude  longitude  elevation  wind_direction  \
      0     MSP 2009-01-01 00:00:00   44.8831   -93.2289      265.8           140.0
      1     MSP 2009-01-01 00:53:00   44.8831   -93.2289      265.8           140.0
      2     MSP 2009-01-01 01:53:00   44.8831   -93.2289      265.8           150.0
      3     MSP 2009-01-01 02:29:00   44.8831   -93.2289      265.8           140.0
      4     MSP 2009-01-01 02:53:00   44.8831   -93.2289      265.8           160.0

        wind_type  wind_speed  ceiling_height ceiling_det_code celing_CAVOK  \
      0         N        36.0     3309.428571                9            N
      1         N        36.0     3658.000000                M            N
      2         V        31.0     3048.000000                M            N
      3         N        62.0     3353.000000                M            N
      4         N        46.0     3048.000000                M            N

         visibility_dist visibility_variability  air_temparature  \
      0          16000.0                      N           -156.0
      1          16093.0                      N           -156.0
      2          16093.0                      N           -133.0
      3          16093.0                      N           -130.0
      4          16093.0                      N           -128.0

         dew_point_temparature  sea_level_pressure  UNIX_TIMESTAMP
      0                 -211.0             10274.0      1230768000
      1                 -206.0             10255.0      1230771180
      2                 -194.0             10237.0      1230774780
```

```
3                 -190.0       10204.8     1230776940
4                 -189.0       10217.0     1230778380
```

[45]:

```
def binary_search_nearest_rows(data, target_timestamp, num_neighbors=3):
    """
    Perform binary search to find the nearest rows in a DataFrame for a given␣
 ↪timestamp.

    Parameters:
        data (DataFrame): Input DataFrame with timestamps.
        target_timestamp (int): Target timestamp for which the nearest rows␣
 ↪need to be found.
        num_neighbors (int): Number of nearest rows to find on each side of the␣
 ↪target timestamp.

    Returns:
        DataFrame: DataFrame containing the rows corresponding to the nearest␣
 ↪timestamps.
    """
    # Sort DataFrame by the timestamp column
    data_sorted = data.sort_values(by='UNIX_TIMESTAMP')

    # Convert the sorted timestamps column to a list
    timestamps = data_sorted['UNIX_TIMESTAMP'].tolist()

    # Binary search to find the nearest timestamp
    low = 0
    high = len(timestamps) - 1

    while low <= high:
        mid = (low + high) // 2
        mid_timestamp = timestamps[mid]

        if mid_timestamp == target_timestamp:
            nearest_indices = [mid]
            break

        elif mid_timestamp < target_timestamp:
            low = mid + 1

        else:
            high = mid - 1

    else:
        # Find the nearest timestamp
        nearest_indices = []
        if high < 0:
            nearest_indices.append(low)
        elif low >= len(timestamps):
            nearest_indices.append(high)
        else:
            if abs(timestamps[low] - target_timestamp) < abs(timestamps[high] -␣
↪target_timestamp):
                nearest_indices.append(low)
            else:
                nearest_indices.append(high)

    # Ensure we have enough neighbors
    while len(nearest_indices) < num_neighbors:
        if nearest_indices[0] - 1 >= 0:
            nearest_indices.insert(0, nearest_indices[0] - 1)
        else:
            nearest_indices.append(nearest_indices[-1] + 1)

    # Extract the rows corresponding to the nearest timestamps
    nearest_rows = data_sorted.iloc[nearest_indices]

    return nearest_rows
```

[46]:

```
binary_search_nearest_rows(weather1, 1262346720)[['latitude', 'longitude',␣
↪'elevation',
       'wind_direction', 'wind_type', 'wind_speed', 'ceiling_height',
       'ceiling_det_code', 'celing_CAVOK', 'visibility_dist',
       'visibility_variability', 'air_temparature', 'dew_point_temparature',
       'sea_level_pressure', 'UNIX_TIMESTAMP']]
```

[46]:

```
       latitude  longitude  elevation  wind_direction wind_type  wind_speed  \
12994   44.8831   -93.2289      265.8           300.0         N        31.0
12995   44.8831   -93.2289      265.8           320.0         N        36.0
12996   44.8831   -93.2289      265.8           310.0         N        21.0

       ceiling_height ceiling_det_code celing_CAVOK  visibility_dist  \
12994           366.0                M            N          16093.0
12995           610.0                M            N          16093.0
12996           701.0                M            N          16093.0

      visibility_variability  air_temparature  dew_point_temparature  \
12994                      N           -170.0                 -210.0
12995                      N           -170.0                 -210.0
12996                      N           -170.0                 -210.0

       sea_level_pressure  UNIX_TIMESTAMP
12994        10302.932552      1262345460
12995        10306.167846      1262345880
```

```
12996        10309.400485      1262346660
```

2 FINAL DATA PREP

[48]:

```python
import tqdm
all_weather = {}
all_rows = []
SYR_DF = pd.read_csv(weather_data_path.format('SYR'), parse_dates=['date'])
SYR_DF['UNIX_TIMESTAMP'] = SYR_DF['date'].apply(lambda x: int(x.timestamp()))
for index, row in tqdm.tqdm_notebook(data.iterrows(), total=len(data)):
    station = row['Origin Airport']
    if station not in all_weather:
        all_weather[station] = pd.read_csv(weather_data_path.format(station),␣
 ↪parse_dates=['date'])
        all_weather[station]['UNIX_TIMESTAMP'] = all_weather[station]['date'].
 ↪apply(lambda x: int(x.timestamp()))
    weatherDF = all_weather[station]
    row = row.to_dict()
    row_unix_ts = row['UNIX_TIMESTAMP']

    closest_rows = binary_search_nearest_rows(weatherDF, row_unix_ts,␣
 ↪num_neighbors=5)[['latitude', 'longitude', 'elevation',
       'wind_direction', 'wind_type', 'wind_speed', 'ceiling_height',
       'ceiling_det_code', 'celing_CAVOK', 'visibility_dist',
       'visibility_variability', 'air_temparature', 'dew_point_temparature',
       'sea_level_pressure']]

    for nc in ['latitude', 'longitude', 'elevation',
       'wind_direction', 'wind_type', 'wind_speed', 'ceiling_height',
       'ceiling_det_code', 'celing_CAVOK', 'visibility_dist',
       'visibility_variability', 'air_temparature', 'dew_point_temparature',
       'sea_level_pressure']:
        if nc in ['wind_type',  'ceiling_det_code', 'celing_CAVOK',␣
 ↪'visibility_variability']:
            row[nc] = closest_rows[nc].mode().iloc[0]
        else:
            row[nc] = closest_rows[nc].mean()

    closest_rows = binary_search_nearest_rows(SYR_DF, row_unix_ts,␣
 ↪num_neighbors=5)[['latitude', 'longitude', 'elevation',
       'wind_direction', 'wind_type', 'wind_speed', 'ceiling_height',
       'ceiling_det_code', 'celing_CAVOK', 'visibility_dist',
       'visibility_variability', 'air_temparature', 'dew_point_temparature',
       'sea_level_pressure']]

    for nc in ['latitude', 'longitude', 'elevation',
       'wind_direction', 'wind_type', 'wind_speed', 'ceiling_height',
       'ceiling_det_code', 'celing_CAVOK', 'visibility_dist',
       'visibility_variability', 'air_temparature', 'dew_point_temparature',
       'sea_level_pressure']:
        if nc in ['wind_type',  'ceiling_det_code', 'celing_CAVOK',␣
 ↪'visibility_variability']:
            row['SYR_'+nc] = closest_rows[nc].mode().iloc[0]
        else:
            row['SYR_'+nc] = closest_rows[nc].mean()
    all_rows.append(row)
new_data_df = pd.DataFrame(all_rows, index=data.index)
```

```
/tmp/ipykernel_32272/3525575574.py:6: TqdmDeprecationWarning: This function will
be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  for index, row in tqdm.tqdm_notebook(data.iterrows(), total=len(data)):

  0%|          | 0/113671 [00:00<?, ?it/s]
```

[49]:

```
new_data_df.head()
```

[49]:

```
  Carrier Code Date (MM/DD/YYYY) Origin Airport Scheduled Arrival Time  \
0           B6        2010-01-01            JFK                  00:01
1           B6        2010-01-01            JFK                  08:55
2           MQ        2010-01-01            ORD                  11:20
3           9E        2010-01-01            DTW                  11:44
4           B6        2010-01-01            JFK                  11:52

   Scheduled Elapsed Time (Minutes) FLIGHT_STATUS  month  day  season WeekDay  \
0                                76          LATE      1    1  winter  Friday
1                                75          LATE      1    1  winter  Friday
2                               100        ONTIME      1    1  winter  Friday
3                                84          LATE      1    1  winter  Friday
4                                71          LATE      1    1  winter  Friday

   … SYR_wind_type  SYR_wind_speed  SYR_ceiling_height  \
0  …             N            17.4          741.370199
1  …             C             0.0          548.800000
2  …             C             0.0          841.400000
3  …             C             0.0         1006.000000
4  …             C             0.0         1006.000000

   SYR_ceiling_det_code  SYR_celing_CAVOK  SYR_visibility_dist  \
0                     M                 N               6110.0
1                     M                 N               2414.0
2                     M                 N               3138.4
3                     M                 N               3138.4
4                     M                 N               3138.4
```

```
  SYR_visibility_variability  SYR_air_temparature  SYR_dew_point_temparature  \
0                          N                 -4.8                      -24.0
1                          N                -17.0                      -28.8
2                          N                -19.4                      -29.6
3                          N                -19.8                      -29.2
4                          N                -19.8                      -29.2

  SYR_sea_level_pressure
0           10154.572817
1           10132.352069
2           10129.796558
3           10129.482076
4           10129.482076

[5 rows x 40 columns]
```

[50]:

```
new_data_df.isna().sum()
```

[50]:

```
Carrier Code                          0
Date (MM/DD/YYYY)                     0
Origin Airport                        0
Scheduled Arrival Time                0
Scheduled Elapsed Time (Minutes)      0
FLIGHT_STATUS                         0
month                                 0
day                                   0
season                                0
WeekDay                               0
UNIX_DATE                             0
UNIX_TIMESTAMP                        0
latitude                              0
longitude                             0
elevation                             0
wind_direction                        0
wind_type                             0
wind_speed                            0
ceiling_height                        0
ceiling_det_code                      0
celing_CAVOK                          0
visibility_dist                       0
visibility_variability                0
air_temparature                       0
dew_point_temparature                 0
sea_level_pressure                  728
SYR_latitude                          0
SYR_longitude                         0
```

```
SYR_elevation                         0
SYR_wind_direction                    0
SYR_wind_type                         0
SYR_wind_speed                        0
SYR_ceiling_height                    0
SYR_ceiling_det_code                  0
SYR_celing_CAVOK                      0
SYR_visibility_dist                   0
SYR_visibility_variability            0
SYR_air_temparature                   0
SYR_dew_point_temparature             0
SYR_sea_level_pressure                0
dtype: int64
```

```
[51]: new_data_df['sea_level_pressure'] = new_data_df['sea_level_pressure'].
       ↪fillna(new_data_df['sea_level_pressure'].mean())
```

```
[52]: new_data_df.isna().sum()
```

```
[52]: Carrier Code                        0
      Date (MM/DD/YYYY)                   0
      Origin Airport                      0
      Scheduled Arrival Time              0
      Scheduled Elapsed Time (Minutes)    0
      FLIGHT_STATUS                       0
      month                               0
      day                                 0
      season                              0
      WeekDay                             0
      UNIX_DATE                           0
      UNIX_TIMESTAMP                      0
      latitude                            0
      longitude                           0
      elevation                           0
      wind_direction                      0
      wind_type                           0
      wind_speed                          0
      ceiling_height                      0
      ceiling_det_code                    0
      celing_CAVOK                        0
      visibility_dist                     0
      visibility_variability              0
      air_temparature                     0
      dew_point_temparature               0
      sea_level_pressure                  0
      SYR_latitude                        0
      SYR_longitude                       0
```

```
SYR_elevation                       0
SYR_wind_direction                  0
SYR_wind_type                       0
SYR_wind_speed                      0
SYR_ceiling_height                  0
SYR_ceiling_det_code                0
SYR_celing_CAVOK                    0
SYR_visibility_dist                 0
SYR_visibility_variability          0
SYR_air_temparature                 0
SYR_dew_point_temparature           0
SYR_sea_level_pressure              0
dtype: int64
```

[60]:

```
# new_data_df.to_csv("SYR_ORIGIN_WEATHER_IMPUTED.csv", index=False)
```

[61]:

```
def create_data_2nd_model_data_ext(df, num_prev=3):
    extended_data = []
    for index, row in df.iterrows():
        for i in range(num_prev):
            if index - i - 1 >= 0:
                extended_row = row.copy()  # Create a copy of the current row
                extended_row['PREV_STAT'] = df.iloc[index - i -␣
  ↪1]['FLIGHT_STATUS']
                extended_data.append(extended_row)
            else:
                extended_row = row.copy()  # Create a copy of the current row
                extended_row['PREV_STAT'] = 'ONTIME'
                extended_data.append(extended_row)
    df_extended = pd.DataFrame(extended_data)
    return df_extended.reset_index(drop=True)
```

[62]:

```
create_data_2nd_model_data_ext(new_data_df).to_csv("SYR_ORIGIN_3HOP.csv",␣
  ↪index=False)
```

[63]:

```
create_data_2nd_model_data_ext(new_data_df, 1).to_csv("SYR_ORIGIN_1HOP.csv",␣
  ↪index=False)
```

[64]:

```
create_data_2nd_model_data_ext(new_data_df, 2).to_csv("SYR_ORIGIN_2HOP.csv",␣
  ↪index=False)
```

[ ]: