2-Vet_Clinic_Wait_Times_Data_Prep

April 27, 2021

1 File Information

Name: Amie Davis

Course: DSC680 - Data Science

Assignment: Project2 - Vet Clinic Wait Times

Purpose: Create single dataset for visualizations

Usage: Python 3.7.6

Developed using Jupter Notebook 6.0.3

2 Data Source

Proprietary data provided by DoveLewis Animal Hospital, Portland, OR

3 Part 2

In Part 2, I will create a single dataset for visualization, further analysis, and modeling

3.1 Import required packages

[1]:
```
# Suppress Warnings
import warnings
warnings.filterwarnings('ignore')

import pandas as pd
import numpy as np
import csv
```

4 Prepare Data

[2]:
```
# Load data into pandas dataframes

# Patient Data
patient_file = "Data\patient_new.csv"
patient_df = pd.read_csv(patient_file)
```

```
# Whiteboard Data
board_file = "Data\WhiteBoard Tracker.xlsx"
board_df = pd.read_excel(board_file)
```

[3]:
```
# Remove irrelevant and redundant fields

# Drop unneeded columns
board_df.drop(['Time', 'TIME Hour',
               'Unnamed: 11', 'Unnamed: 12', 'KEY:',
       'Each row on this spreadsheet is a 5 minute snapshot of how many␣
 ↪patients we have in the hospital in our two main departments (Emergency Room␣
 ↪and ICU) at the respective time . This is the tool we use to see what times␣
 ↪of the day had the highest patient loads after the fact, not in real time.',
       'Unnamed: 15', 'Unnamed: 16', 'Unnamed: 17', 'Unnamed: 18',
       'Unnamed: 19', 'Unnamed: 20', 'Unnamed: 21'],
        axis=1, inplace = True)

patient_df.drop(['Unnamed: 0'],
        axis=1, inplace = True)

# Verify Change
#print(patient_df.columns)
#print(board_df.columns)
```

[4]:
```
# Convert Appt Date to datetime
patient_df['Appt_Date'] = patient_df['Appt_Date'].astype('datetime64')
patient_df['Appointment Date1'] = patient_df['Appointment Date1'].
 ↪astype('datetime64')
patient_df['Appointment Date2'] = patient_df['Appointment Date2'].
 ↪astype('datetime64')
patient_df['Appointment Date3'] = patient_df['Appointment Date3'].
 ↪astype('datetime64')

print(board_df.dtypes)
print(patient_df.dtypes)
```

```
Row ID                          int64
Outpatient Count                int64
ICU Patient Count               int64
Time Stamp             datetime64[ns]
Weekday                datetime64[ns]
Date                   datetime64[ns]
Year                            int64
Week                            int64
Month                           int64
dtype: object
```

```
Department                          object
Consult Date                        object
Consult Division                    object
Clinical Number                      int64
Patient Number                       int64
Triage Type                        float64
Clinical Description                object
Appointment Type1                   object
Appointment Type2                   object
Appointment Type3                   object
Appointment Date1         datetime64[ns]
Appointment Date2         datetime64[ns]
Appointment Date3         datetime64[ns]
Presenting Problem1                 object
Presenting Problem2                 object
Presenting Problem3                 object
Therapeutic-Procedure1              object
Therapeutic-Procedure2              object
Therapeutic-Procedure3              object
Therapeutic-Procedure4              object
Therapeutic-Procedure5              object
Therapeutic-Procedure6              object
Therapeutic-Procedure7              object
Therapeutic-Procedure8              object
Appt_Date                 datetime64[ns]
dtype: object
```

[5]:
```
# Create derived columns to join datasets
# Convert to timedelta since 1st record 2Jul2018

# Split whiteboard timestamp into start and end of time window
board_df['flt_start_time'] = (board_df['Time Stamp'] - np.
 ↪datetime64('2018-07-02','D')) / np.timedelta64(1,'D')
board_df['flt_end_time'] = (board_df['Time Stamp'] - np.
 ↪datetime64('2018-07-02','D') + np.timedelta64(5,'m')) / np.timedelta64(1,'D')
board_df['board_start'] = round(board_df['flt_start_time'],3)

patient_df['flt_appt_time'] = (patient_df['Appt_Date'] - np.
 ↪datetime64('2018-07-02','D')) / np.timedelta64(1,'D')

#board_df.head()
#patient_df.head()
```

[6]:
```
# Load board times into list to match against patient data
board_time_lst = board_df['board_start'].tolist()
board_time_lst.sort()    #ensure times are ordered
```

```
# Get the earliest whiteboard start time
min_board_flt = board_df['board_start'].min()
#print(min_board_flt)
```

[7]:
```
# Function to calculate board start time for patient record
def find_board_start(x):

    board_start = 0
    last_board = 183

    # Leave null if patient record older than whiteboard tracker
    if x < min_board_flt:
        board_start = None

    else:

        # Loop through list of board times
        for i in board_time_lst:

            # Use the 1st board start time greater than the appt_time
            if x >= last_board and x < i:
                board_start = last_board

                break

            # Save last board_start from the list for next interation
            last_board = i

    return board_start

patient_df['board_start']=patient_df['flt_appt_time'].apply(find_board_start)
#patient_df.tail(50)
```

[8]:
```
# Join datasets
joint_df = pd.merge(patient_df, board_df, how='left', on='board_start')
```

[9]:
```
# Export detailed patient dataset for use in Visualizations
joint_df.to_csv(r'Data\joined_data.csv', index = False, header=True)
```

[10]:
```
# Prepare data for modeling
# Only look at data from 2021 to include Triage Types and Urgent Care␣
 ↪implementation
group_df = joint_df[joint_df.Year == 2021]

# Replace null values with text in order to include records in count
group_df = group_df.fillna('EMPTY')
```

```
# Pull Hour from board Timestamp to treat as categorical, along with month,␣
 ↪week, and weekday
group_df['Hour'] = group_df['Time Stamp'].dt.hour

# Group by whiteboard times to obtain patient counts during the window
# Group by categorical fields and convert to pandas df
group_df = group_df.groupby([
    'Row ID',
    'Time Stamp',
    'Month',
    'Week',
    'Weekday',
    'Hour',
    'Outpatient Count',
    'ICU Patient Count',
    'Department',
    'Triage Type'
]).size().to_frame('size').reset_index()

group_df.tail()
```

[10]:

	Row ID	Time Stamp	Month	Week	Weekday	Hour	Outpatient Count	ICU Patient Count	Department	Triage Type	size
8444	435295.0	2021-04-11 22:14:47	4.0	16.0	1900-01-01	22	23.0	12.0	A-ECC	EMPTY	1
8445	435297.0	2021-04-11 22:24:51	4.0	16.0	1900-01-01	22	27.0	12.0	A-ECC	EMPTY	1
8446	435298.0	2021-04-11 22:29:54	4.0	16.0	1900-01-01	22	27.0	13.0	A-ECC	EMPTY	2
8447	435304.0	2021-04-11 23:00:13	4.0	16.0	1900-01-01	23	28.0	13.0	A-ECC	EMPTY	1
8448	435308.0	2021-04-11 23:20:22	4.0	16.0	1900-01-01	23	27.0	14.0	A-ECC	EMPTY	1

[11]:
```
# Export grouped dataset for use in Modeling
group_df.to_csv(r'Data\grouped_data.csv', index = False, header=True)
```

[12]:
```
# Function to get ECC patient counts for each board record
def get_ecc_cnt(x):

    # Given Row ID, count ECC patients
    df = group_df.loc[(group_df['Row ID'] == x) & (group_df['Department'] ==␣
 ↪'A-ECC')]
    df = df.groupby(['Department']).sum()

    if df.empty:
```

```
        rec_cnt = 0
    else:
        rec_cnt = df['size'].item()

    return rec_cnt
```

[13]:
```
# Function to get Cardiology patient counts for each board record
def get_cardio_cnt(x):

    # Given Row ID, count Cardiology patients
    df = group_df.loc[(group_df['Row ID'] == x) & (group_df['Department'] ==␣
 ↪'B-CARDIO')]
    df = df.groupby(['Department']).sum()

    if df.empty:
        rec_cnt = 0
    else:
        rec_cnt = df['size'].item()

    return rec_cnt
```

[14]:
```
# Function to get Internal Medicine patient counts for each board record
def get_im_cnt(x):

    # Given Row ID, count Internal Medicine patients
    df = group_df.loc[(group_df['Row ID'] == x) & (group_df['Department'] ==␣
 ↪'C-IM')]
    df = df.groupby(['Department']).sum()

    if df.empty:
        rec_cnt = 0
    else:
        rec_cnt = df['size'].item()

    return rec_cnt
```

[15]:
```
# Function to get URGENT CARE patient counts for each board record
def get_uc_cnt(x):

    # Given Row ID, count Urgent Care patients
    df = group_df.loc[(group_df['Row ID'] == x) & (group_df['Department'] ==␣
 ↪'D-URGENT CARE')]
    df = df.groupby(['Department']).sum()

    if df.empty:
        rec_cnt = 0
    else:
```

```
        rec_cnt = df['size'].item()

    return rec_cnt
```

[16]:
```
# Function to get Abandoned patient counts for each board record
def get_aband_cnt(x):

    # Given Row ID, count Abandoned patients
    df = group_df.loc[(group_df['Row ID'] == x) & (group_df['Department'] ==␣
 ↪'NO EXAM')]
    df = df.groupby(['Department']).sum()

    if df.empty:
        rec_cnt = 0
    else:
        rec_cnt = df['size'].item()

    return rec_cnt
```

[17]:
```
# Function to get STAT patient counts for each board record
def get_stat_cnt(x):

    # Given Row ID, count STAT patients
    df = group_df.loc[(group_df['Row ID'] == x) & (group_df['Triage Type'] ==␣
 ↪1)]
    df = df.groupby(['Triage Type']).sum()

    if df.empty:
        rec_cnt = 0
    else:
        rec_cnt = df['size'].item()

    return rec_cnt
```

[18]:
```
# Function to get URGENT patient counts for each board record
def get_urg_cnt(x):

    # Given Row ID, count STAT patients
    df = group_df.loc[(group_df['Row ID'] == x) & (group_df['Triage Type'] ==␣
 ↪2)]
    df = df.groupby(['Triage Type']).sum()

    if df.empty:
        rec_cnt = 0
    else:
        rec_cnt = df['size'].item()
```

```
    return rec_cnt
```

[19]:
```
# Function to get Stable Emergency patient counts for each board record
def get_stab_cnt(x):

    # Given Row ID, count Stable Emergency patients
    df = group_df.loc[(group_df['Row ID'] == x) & (group_df['Triage Type'] ==␣
 ↪3)]
    df = df.groupby(['Triage Type']).sum()

    if df.empty:
        rec_cnt = 0
    else:
        rec_cnt = df['size'].item()

    return rec_cnt
```

[20]:
```
# Function to get Urgent Care candidate counts for each board record
def get_uc_cnt(x):

    # Given Row ID, count Urgent Care candidates
    df = group_df.loc[(group_df['Row ID'] == x) & (group_df['Triage Type'] ==␣
 ↪4)]
    df = df.groupby(['Triage Type']).sum()

    if df.empty:
        rec_cnt = 0
    else:
        rec_cnt = df['size'].item()

    return rec_cnt
```

[21]:
```
# Function to get counts for Put-to-Sleep (PTS) patients each board record
def get_pts_cnt(x):

    # Given Row ID, count PTS patients
    df = group_df.loc[(group_df['Row ID'] == x) & (group_df['Triage Type'] ==␣
 ↪5)]
    df = df.groupby(['Triage Type']).sum()

    if df.empty:
        rec_cnt = 0
    else:
        rec_cnt = df['size'].item()

    return rec_cnt
```

[22]:
```
# Function to get counts for Non-Urgent patients each board record
def get_non_cnt(x):

    # Given Row ID, count Non-Urgent patients
    df = group_df.loc[(group_df['Row ID'] == x) & (group_df['Triage Type'] ==␣
 ↪6)]
    df = df.groupby(['Triage Type']).sum()

    if df.empty:
        rec_cnt = 0
    else:
        rec_cnt = df['size'].item()

    return rec_cnt
```

[23]:
```
# TESt 408474
df = group_df.loc[(group_df['Row ID'] == 408474) & (group_df['Department'] ==␣
 ↪'B-CARDIO')]
df = df.groupby(['Department']).sum()
df.head()
```

[23]:

Department	Row ID	Month	Week	Hour	Outpatient Count	ICU Patient Count	size
B-CARDIO	408474.0	1.0	2.0	11	11.0	18.0	1

[24]:
```
# Create modeling dataset
model_board_df = board_df[board_df.Year == 2021]

# Get patient counts by department
model_board_df['ecc_dept_cnt'] = model_board_df['Row ID'].apply(get_ecc_cnt)
model_board_df['cardio_dept_cnt'] = model_board_df['Row ID'].
 ↪apply(get_cardio_cnt)
model_board_df['im_dept_cnt'] = model_board_df['Row ID'].apply(get_im_cnt)
model_board_df['uc_dept_cnt'] = model_board_df['Row ID'].apply(get_uc_cnt)
model_board_df['aband_cnt'] = model_board_df['Row ID'].apply(get_aband_cnt)

# Get patient counts by triage type
model_board_df['stat_tri_cnt'] = model_board_df['Row ID'].apply(get_stat_cnt)
model_board_df['urg_tri_cnt'] = model_board_df['Row ID'].apply(get_urg_cnt)
model_board_df['stab_tri_cnt'] = model_board_df['Row ID'].apply(get_stab_cnt)
model_board_df['uc_tri_cnt'] = model_board_df['Row ID'].apply(get_uc_cnt)
model_board_df['pts_tri_cnt'] = model_board_df['Row ID'].apply(get_pts_cnt)
model_board_df['non_tri_cnt'] = model_board_df['Row ID'].apply(get_non_cnt)
```

[25]:
```
# Add datetime categories (month, week, and weekday)
model_board_df['Hour'] = model_board_df['Time Stamp'].dt.hour
model_board_df['Month'] = model_board_df['Time Stamp'].dt.month
model_board_df['Week_No'] = model_board_df['Time Stamp'].dt.weekofyear
model_board_df['Weekday'] = model_board_df['Time Stamp'].dt.dayofweek

# Drop unneeded columns
model_board_df.drop(['flt_start_time','flt_end_time','board_start',
              'Date','Year','Week'],
        axis=1, inplace = True)
```

[26]:
```
model_board_df.head()
```

[26]:

	Row ID	Outpatient Count	ICU Patient Count	Time Stamp	\
152296	406905	14	23	2021-01-01 00:02:39	
152297	406906	14	23	2021-01-01 00:07:41	
152298	406907	13	23	2021-01-01 00:12:43	
152299	406908	13	23	2021-01-01 00:17:45	
152300	406909	12	23	2021-01-01 00:22:47	

	Weekday	Month	ecc_dept_cnt	cardio_dept_cnt	im_dept_cnt	\
152296	4	1	0	0	0	
152297	4	1	0	0	0	
152298	4	1	0	0	0	
152299	4	1	0	0	0	
152300	4	1	1	0	0	

	uc_dept_cnt	aband_cnt	stat_tri_cnt	urg_tri_cnt	stab_tri_cnt	\
152296	0	0	0	0	0	
152297	0	0	0	0	0	
152298	0	0	0	0	0	
152299	0	0	0	0	0	
152300	0	0	0	0	0	

	uc_tri_cnt	pts_tri_cnt	non_tri_cnt	Hour	Week_No
152296	0	0	0	0	53
152297	0	0	0	0	53
152298	0	0	0	0	53
152299	0	0	0	0	53
152300	0	0	0	0	53

[27]:
```
# Export whiteboard window dataset for use in Modeling
model_board_df.to_csv(r'Data\modeling_data.csv', index = False, header=True)
```