Reproducible_Reporting

July 8, 2021

1 Reproducible Reporting using Markdown - Python

1.1 Setup

1.1.1 Import packages

[ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
import statsmodels.formula.api as smf

# Remove unrelated warning
import warnings
warnings.simplefilter(action="ignore", category=FutureWarning)
```

1.1.2 Download OASIS Dataset

We'll be downloading the data from `http://www.oasis-brains.org/pdf/oasis_longitudinal.csv`.

A full explanation of the data can be found in:

Open Access Series of Imaging Studies (OASIS): Longitudinal MRI Data in Nondemented and Demented Older Adults. Marcus, DS, Fotenos, AF, Csernansky, JG, Morris, JC, Buckner, RL, 2010. Journal of Cognitive Neuroscience, 22, 2677-2684. doi: 10.1162/jocn.2009.21407

[ ]:
```
df = pd.read_csv('http://www.oasis-brains.org/pdf/oasis_longitudinal.csv')
# Print out the top 5 rows of the DataFrame
df.head(5)
```

1.2 Pre-processing the Data

1.2.1 Dementia Status from CDR

We're told in the associated paper (linked above) that the CDR values correspond to the following levels of severity: * 0 = No dementia * 0.5 = Very mild Dementia * 1 = Mild Dementia * 2 = Moderate Dementia * 3 = Severe Dementia

So let's derive this "Status" from the CDR values:

[ ]:
```
# Create a mapper from values to status/severity
cdr_mapper = {
    0: "No Dementia",
    0.5: "Very Mild Dementia",
    1: "Mild Dementia",
    2: "Moderate Dementia",
    3: "Severe Dementia",
}
# Create a column, mapping CDR values to this status
# Convert to categorical type, with severity as order
df["Status"] = pd.Categorical(
    df["CDR"].map(cdr_mapper),
    categories=cdr_mapper.values(),
    ordered=True
)
```

[ ]:
```
# Create colours for the different 'Status' categories
status_colours = {
    i:j for i,j in zip(cdr_mapper.values(), sns.color_palette("colorblind", 5))
}
```

Now we can look at this status, and how many records there are both overall, and at each of the visits (or time points).

[ ]:
```
# Let's look at how many records we have of each severity
df["Status"].value_counts()
```

[ ]:
```
# There are no records for "Severe Dementia", so we can tidy things up by␣
 ↪removing this unused category
df["Status"] = df["Status"].cat.remove_unused_categories()
```

[ ]:
```
# How many of each status do we have at each visit?
df.groupby(["Visit", "Status"]).size()
```

1.2.2 Variable Types

[ ]:
```
df.dtypes
```

Note: Further coercing of types (e.g. "Visit" is ordinal) is good practice, but the necessity depends on your analysis.

1.2.3 Convert Categorical Data

[ ]:
```
df["Sex"] = df["M/F"].map({
  "M": 0,
  "F": 1
})
```

1.2.4 Visits & Baseline Data

[ ]:
```
df["Visit"].value_counts()
```

Filter for baseline data, and look at their status on arrival

[ ]:
```
df_baseline = df.loc[df["Visit"]==1]
```

[ ]:
```
df_baseline["Status"].value_counts()
```

1.3 Data Exploration & Analysis

1.3.1 Summarize Data

[ ]:
```
df.describe(include="all")
```

1.3.2 Missing Data

[ ]:
```
# Count amount of missing data
df.isna().sum(0)
```

1.3.3 Visualize Data

[ ]:
```
# Histograms
fig, axes = plt.subplots(2, 2, figsize=(12,8))

axes = axes.flatten()

for i, x_col in enumerate(["Age", "EDUC", "MMSE"]):
  sns.histplot(data=df, x=x_col, ax=axes[i])

fig.delaxes(axes[-1])
```

[ ]:
```
# Histograms stratified by dementia status
fig, axes = plt.subplots(2, 2, figsize=(12,8))
# Make it easier to loop over
axes = axes.flatten()
# Loop through the columns we want to plot a histogram for
for i, x_col in enumerate(["Age", "EDUC", "MMSE"]):
    sns.histplot(data=df, x=x_col, hue="Status", multiple="stack", ax=axes[i])
# Manually tidy up the legends
axes[0].get_legend().remove()
axes[1].get_legend().remove()
# Delete the extra axis (as we only plot 3 columns)
fig.delaxes(axes[-1])
```

1.3.4 MMSE trajectories over time

We can visualize the MMSE trajectories of each subject over time to get an overall view of the cohort.

[ ]:
```
fig, ax = plt.subplots(figsize=(12,6))
for _, indiv_data in df.groupby("Subject ID"):
    ax = sns.lineplot(
        data=indiv_data,
        x="MR Delay",
        y="MMSE",
        ax=ax,
        color="grey",
        alpha=0.5,
        linewidth=1.25,
        linestyle="solid"
    )
```

We can make this graph a little more interesting by adding colour to each line, corresponding to the dementia status:

[ ]:
```
fig, ax = plt.subplots(figsize=(12,6))

# Loop over the individual subjects
for group, indiv_data in df.groupby(["Subject ID"]):
    # Extract xy data
    xy = indiv_data[["MR Delay", "MMSE"]].values
    # Plot normally for patients that do not change status
    if len(indiv_data["Status"].unique()) == 1:
      ax.plot(
          xy[:, 0],
          xy[:, 1],
          color=status_colours[indiv_data["Status"].unique()[0]],
          alpha=0.5,
          linewidth=1.25
      )
    else:
      # Duplicate points to construct line segments of different colours
      for i, (start, stop) in enumerate(zip(xy[:-1], xy[1:])):
        x, y = zip(start, stop)
        ax.plot(
            x, y,
            color=status_colours[indiv_data["Status"].iloc[i]],
            alpha=0.5,
            linewidth=1.25
        )
# Add the labels
ax.set_xlabel("MR Delay")
ax.set_ylabel("MMSE")
```

1.3.5 Regression Analysis

Simple Linear Regression

[ ]:
```
# Select the columns to iterate over
x_cols = ["Age", "Sex", "EDUC", "CDR"]
# Choose nicer names to display for each column
x_labels = ["Age", "Sex", "Yrs of Education", "CDR"]

fig, axes = plt.subplots(2,2, figsize=(16,10))

# Select the dependent variable
y = df_baseline["MMSE"].values
# Loop over the columns to fit each regression
for x_col, label, ax in zip(x_cols, x_labels, axes.flatten()):
    # Select x data
    x = df_baseline[x_col].values
    # Create the model
    model = sm.OLS(
        y, sm.add_constant(x)
    )
    # Fit the model
    res = model.fit()
    # Add a plot with regression line
    ax.plot(x, res.predict())
    # and the raw data
    ax.scatter(x, y, c=[status_colours[i] for i in df_baseline["Status"]])
    ax.set_title(label)
    ax.set_ylabel("MMSE")
    # Print the R^2
    print(f"{label} vs MMSE: R-squared = {res.rsquared:.3f}")
```

Then we can do the same again, but this time stratify by the dementia status (not including CDR this time):

[ ]:
```
x_cols = ["Age", "Sex", "EDUC"]
x_labels = ["Age", "Sex", "Yrs of Education"]

fig, axes = plt.subplots(2,2, figsize=(16,10))
axes = axes.flatten()

for x_col, label, ax in zip(x_cols, x_labels, axes):
    for name, group_data in df_baseline.groupby("Status", observed=True):
        y = group_data["MMSE"]
        x = group_data[x_col]
        model = sm.OLS(
            y, sm.add_constant(x)
        )
        res = model.fit()
```

```
        ax.plot(x, res.predict())
        ax.scatter(x, y, color=status_colours[name])
    ax.set_title(label)
    ax.set_ylabel("MMSE")
# Delete the extra axis (as we only plot 3 columns)
fig.delaxes(axes[-1])
```

Multiple Linear Regression We can create a regression model using age, sex, education, and CDR as our independent variables together:

[ ]:
```
# Create string for our independent variables
x = "Age + EDUC + Sex + C(CDR)"
# And select the dependent one
y = "MMSE"

mod = smf.ols(formula = f"{y} ~ {x}", data=df_baseline)
res = mod.fit()

# Print out a nice summary of results
print(res.summary())
```

[ ]:
```
# Call statsmodel function for partial regression plot
fig = sm.graphics.plot_partregress_grid(res)
# Making the output nicer
fig.set_figheight(10)
fig.set_figwidth(12)
fig.tight_layout(pad=3)
```

1.3.6 Results Table

Either using some of the analysis above, or through additional analysis of your own, create a summary results table below. What it includes is up to you, the overall aim is to get familiar with creating tables in markdown.

1.4 Export to PDF

Now we can export this report to PDF, using the steps below.

Note: You may need to change the paths and filenames depending your Google Drive structure, and what you have named this notebook.

[ ]:
```
# Install what we need to create the PDF
%%capture
!apt update
!apt install texlive-xetex texlive-fonts-recommended texlive-generic-recommended
```

[ ]:
```
# Mount Google Drive
from google.colab import drive
```

```
drive.mount('/content/gdrive')
```

[ ]:
```
# Create a DEMON folder and move this notebook into it
%%bash
mkdir -p gdrive/MyDrive/DEMON/
if [ -f gdrive/MyDrive/Reproducible_Reporting.ipynb ]; then
  mv gdrive/MyDrive/Reproducible_Reporting.ipynb gdrive/MyDrive/DEMON/
 ↪Reproducible_Reporting.ipynb
fi
ls gdrive/MyDrive/DEMON/
```

[ ]:
```
# Change to the drive we created above
%cd gdrive/MyDrive/DEMON/
```

[ ]:
```
!jupyter nbconvert Reproducible_Reporting.ipynb --to pdf --output ./
 ↪Reproducible_Reporting.pdf
```