contextuaLearningColorCue

May 9, 2024

[ ]:
```python
import io
from itertools import groupby
import json
import os
import re
import warnings
from collections import defaultdict
from datetime import datetime
from os import listdir

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import researchpy as rp
import seaborn as sns
from seaborn.relational import scatterplot
import statsmodels.api as sm
import statsmodels.formula.api as smf
from frites import set_mpl_style
from scipy import stats
from scipy.stats import kruskal, linregress, normaltest, pearsonr
from statsmodels.formula.api import ols
from statsmodels.stats.diagnostic import het_white

set_mpl_style()
```

[ ]:
```python
def process_events(rows, blocks, colnames):
    # If no data, create empty dataframe w/ all cols and types
    if len(rows) == 0:
        rows = ["", ""]
        blocks = []
    # Parse data, dropping useless first column
    if len(rows) == 1:
        list(rows).append("")
    # first col is event type, which we drop later
    colnames = ["type"] + colnames
    coltypes = get_coltypes(colnames)
    df = pd.read_csv(
```

```
        io.StringIO("\n".join(rows)),
        delimiter="\s+",
        header=None,
        names=colnames,
        na_values=".",
        index_col=False,
    )
    df = df.iloc[:, 1:]  # drop the first column
    # Move eye column to end & make factor, append block numbers to beginning␣
↪of data frame
    if "eye" in colnames:
        df = df.iloc[:, [1] + list(range(2, df.shape[1])) + [0]]
        df["eye"] = pd.Categorical(df["eye"], categories=["L", "R"],␣
↪ordered=False)
    df.insert(loc=0, column="trial", value=blocks)
    return df


def process_saccades(saccades, blocks, info):
    sacc_df = process_events(saccades, blocks, get_sacc_header(info))
    # Set amplitudes for any saccades missing start/end coords to NAs because␣
↪they're wonky
    ampl_cols = [col for col in sacc_df.columns if re.search(r"ampl\d*$", col)]
    partial = sacc_df["sxp"].isna() | sacc_df["exp"].isna()
    if any(partial):
        sacc_df.loc[partial, ampl_cols] = pd.NA
    return sacc_df


def process_fixations(fixations, blocks, info):
    return process_events(fixations, blocks, get_fix_header(info))


def process_blinks(blinks, blocks):
    return process_events(blinks, blocks, ["eye", "stime", "etime", "dur"])


def process_messages(msgs, blocks):
    # Process messages from tracker
    msg_mat = [msg.split(" ", 1) for msg in msgs]
    msg_mat = [[msg[0][4:], msg[1].rstrip()] for msg in msg_mat]
    msg_df = pd.DataFrame(msg_mat, columns=["time", "text"])
    msg_df["time"] = pd.to_numeric(msg_df["time"])

    # Append trial numbers to beginning of data frame
    msg_df.insert(0, "trial", blocks)
```

```python
    return msg_df


def process_input(input_data, blocks):
    return process_events(input_data, blocks, ["time", "value"])


def process_buttons(button, blocks):
    return process_events(button, blocks, ["time", "button", "state"])


def from_header(header, field):
    pattern = r"\*\* {}\s*: (.*)".format(re.escape(field))
    matches = [re.findall(pattern, line) for line in header]
    matches = [match for match in matches if match]
    return matches[0][0] if matches else None


def get_resolution(nonsample):
    res = [None, None]
    for pattern in ["DISPLAY_COORDS", "GAZE_COORDS", "RESOLUTION"]:
        display_xy = [s for s in nonsample if pattern in s]
        if len(display_xy) == 0:
            continue
        display_xy = re.sub(f".* {pattern}\\D+(.*)", "\\1", display_xy[0])
        try:
            display_xy = [int(float(s)) for s in display_xy.split()]
        except ValueError:
            continue
        res = [display_xy[2] - display_xy[0] + 1, display_xy[3] - display_xy[1] + 1]
        break
    return res


def get_mount(mount_str):
    # Older EyeLink 1000s may be missing "R" in table mount names, we add one if needed
    if re.search("TABLE$", mount_str):
        mount_str = mount_str + "R"

    mounts = {
        "MTABLER": "Desktop / Monocular / Head Stabilized",
        "BTABLER": "Desktop / Binocular / Head Stabilized",
        "RTABLER": "Desktop / Monocular / Remote",
        "RBTABLER": "Desktop / Binocular / Remote",
        "AMTABLER": "Arm Mount / Monocular / Head Stabilized",
```

```python
        "ARTABLER": "Arm Mount / Monocular / Remote",
        "TOWER": "Tower Mount / Monocular / Head Stabilized",
        "BTOWER": "Tower Mount / Binocular / Head Stabilized",
        "MPRIM": "Primate Mount / Monocular / Head Stabilized",
        "BPRIM": "Primate Mount / Binocular / Head Stabilized",
        "MLRR": "Long-Range Mount / Monocular / Head Stabilized",
        "BLRR": "Long-Range Mount / Binocular / Head Stabilized",
    }

    return mounts[mount_str] if mount_str in mounts else None


def get_raw_header(info):
    eyev = ["xp", "yp", "ps"]

    if not info["mono"]:
        eyev = [f"{e}{s}" for s in ["l", "r"] for e in eyev]

    if info["velocity"]:
        if info["mono"]:
            eyev += ["xv", "yv"]
        else:
            eyev += [f"{e}{s}" for s in ["vl", "vr"] for e in ["x", "y"]]

    if info["resolution"]:
        eyev += ["xr", "yr"]

    if info["input"]:
        eyev += ["input"]

    if info["buttons"]:
        eyev += ["buttons"]

    if info["tracking"]:
        eyev += ["cr.info"]

    if info["htarg"]:
        eyev += ["tx", "ty", "td", "remote.info"]

    return ["time"] + eyev


def get_event_header(info, xy_cols):
    base = ["eye", "stime", "etime", "dur"]

    if info["event.dtype"] == "HREF":
        xy_cols = [f"href.{xy}" for xy in xy_cols] + xy_cols
```

```
    if info["resolution"]:
        xy_cols += ["xr", "yr"]

    return base + xy_cols


def get_sacc_header(info):
    return get_event_header(info, ["sxp", "syp", "exp", "eyp", "ampl", "pv"])


def get_fix_header(info):
    return get_event_header(info, ["axp", "ayp", "aps"])


def get_model(header):
    version_str = from_header(header, "VERSION")
    version_str2 = [x for x in header if re.search("\\*\\* EYELINK II", x)]
    if version_str is None:
        model = "Unknown"
        ver_num = "Unknown"
    elif version_str != "EYELINK II 1":
        model = "EyeLink I"
        ver_num = re.search(r"(\d+.\d+)", version_str).group(1)
    else:
        ver_num = re.search(r"v(\d+.\d+)", version_str2[0]).group(1)
        model = (
            "EyeLink II"
            if float(ver_num) < 2.4
            else "EyeLink 1000"
            if float(ver_num) < 5
            else "EyeLink 1000 Plus"
            if float(ver_num) < 6
            else "EyeLink Portable Duo"
        )
    return [model, ver_num]


def get_coltypes(colnames, float_time=True):
    chr_cols = ["type", "eye", "cr.info", "remote.info"]
    int_cols = ["button", "state", "value"]
    time_cols = ["time", "stime", "etime", "dur"]
    if not float_time:
        int_cols += time_cols

    coltypes = [
        "str" if col in chr_cols else "int64" if col in int_cols else "float64"
```

```
        for col in colnames
    ]

    return coltypes


def get_htarg_regex(binocular):
    htarg_errs = "MANCFTBLRTBLRTBLR" if binocular else "MANCFTBLRTBLR"
    htarg_errs = list(htarg_errs)
    htarg_regex = "(" + "|".join(htarg_errs + ["\\."]) + ")"

    return htarg_regex


def is_float(string):
    return bool(re.search("\\.", string))


def get_info(nonsample, firstcol):
    header = [f for f in nonsample if f.startswith("**")]
    info = {}

    # Get date/time of recording from file
    datetime.strptime(from_header(header, "DATE"), "%a %b %d %H:%M:%S %Y")
    # Get tracker model/version info
    version_info = get_model(header)
    info["model"] = version_info[0]
    info["version"] = version_info[1]

    # Get tracker mount info
    elclcfg = [line for line in nonsample if "ELCLCFG" in line]
    if len(elclcfg) > 0:
        info["mount"] = get_mount(re.findall(r"ELCLCFG\s+(.*)", elclcfg[0])[0])

    # Get display size from file
    screen_res = get_resolution(nonsample)
    info["screen.x"] = screen_res[0]
    info["screen.y"] = screen_res[1]

    # Get pupil size data type (area or diameter)
    pupil_config = [line for i, line in enumerate(nonsample) if firstcol[i] ==␣
↪"PUPIL"]
    if len(pupil_config) > 0:
        info["pupil.dtype"] = pupil_config[-1].split()[1]

    # Find the samples and events config lines in the non-sample input, get␣
↪data types
```

```python
    events_config = [
        line for i, line in enumerate(nonsample) if firstcol[i] == "EVENTS"
    ]
    samples_config = [
        line for i, line in enumerate(nonsample) if firstcol[i] == "SAMPLES"
    ]

    # Find the samples and events config lines in the non-sample input, get data types
    events_config = [
        line for i, line in enumerate(nonsample) if firstcol[i] == "EVENTS"
    ]
    samples_config = [
        line for i, line in enumerate(nonsample) if firstcol[i] == "SAMPLES"
    ]
    if len(events_config) > 0:
        info["event.dtype"] = events_config[-1].split()[1]
    if len(samples_config) > 0:
        info["sample.dtype"] = samples_config[-1].split()[1]

    # Get last config line in file (preferring sample config) and extract remaining info
    config = events_config + samples_config[-1:]
    config = config[-1] if len(config) > 0 else ""
    if config:
        info["sample.rate"] = (
            float(re.findall(r"RATE\s+([0-9]+\.[0-9]+)", config)[0])
            if "RATE" in config
            else None
        )
        info["tracking"] = "\tTRACKING" in config
        info["cr"] = "\tCR" in config
        info["filter.level"] = (
            int(re.findall(r"FILTER\s+([0-9]+)", config)[0])
            if "FILTER" in config
            else None
        )
        info["velocity"] = "\tVEL" in config
        info["resolution"] = "\tRES" in config
        info["htarg"] = "\tHTARG" in config
        info["input"] = "\tINPUT" in config
        info["buttons"] = "\tBUTTONS" in config
        info["left"] = "\tLEFT" in config
        info["right"] = "\tRIGHT" in config
        info["mono"] = not (info["right"] & info["left"])

    return info
```

```python
def process_raw(raw, blocks, info):
    if len(raw) == 0:
        # If no sample data in file, create empty raw DataFrame w/ all␣
↪applicable columns
        raw = ["", ""]
        blocks = pd.Series([], dtype=int)
        colnames = get_raw_header(info)
        coltypes = get_coltypes(colnames, float_time=False)
    else:
        # Determine if timestamps stored as floats (edf2asc option -ftime,␣
↪useful for 2000 Hz)
        float_time = is_float(re.split(r"\s+", raw[0])[0])
        # Generate column names and types based in info in header
        colnames = get_raw_header(info)
        coltypes = get_coltypes(colnames, float_time)
        # Discard any rows with too many or too few columns (usually rows where␣
↪eye is missing)
        row_length = [len(re.split(r"\t", r)) for r in raw]
        med_length = np.median(row_length)
        raw = [r for r, l in zip(raw, row_length) if l == med_length]
        blocks = blocks[row_length == med_length]
        # Verify that generated columns match up with actual maximum row length
        length_diff = med_length - len(colnames)
        # if length_diff > 0:
        #     warnings.warn("Unknown columns in raw data. Assuming first one is␣
↪time, please check the others")
        #     colnames = ["time"] + [f"X{i+1}" for i in range(med_length-1)]
        #     coltypes = "i" + "?"*(med_length-1)
    # Process raw sample data using pandas
    if len(raw) == 1:
        raw.append("")

    raw_df = pd.read_csv(
        io.StringIO("".join(raw)),
        sep="\t",
        header=None,
        names=colnames,
        na_values=np.nan,
        low_memory=False,
    )

    if info["tracking"] and not info["cr"]:
        # Drop CR column when not actually used
        raw_df = raw_df.drop(columns=["cr.info"])
    # Append block numbers to beginning of DataFrame
```

```python
    raw_df.insert(0, "trial", blocks)
    # Replace missing pupil data (zeros) with NaNs
    if "X1" not in raw_df.columns:
        if info["mono"]:
            raw_df.loc[raw_df["ps"] == 0, "ps"] = np.nan
        else:
            raw_df.loc[raw_df["psl"] == 0, "psl"] = np.nan
            raw_df.loc[raw_df["psr"] == 0, "psr"] = np.nan
    return raw_df


def read_asc(fname, samples=True, events=True, parse_all=False):
    with open(fname, "r", encoding="ISO-8859-1", errors="ignore") as f:
        inp = f.readlines()

    # Convert to ASCII
    inp = [line.encode("ascii", "ignore").decode() for line in inp]

    # Get strings prior to first tab for each line for faster string matching
    inp_first = [re.split(r"\s", s)[0] for s in inp]

    # # Get the Trial info for each trial:
    # bias = [
    #     s.split()[4] for s in inp if len(s.split()) > 4 and s.split()[2] == "Trialinfo:"
    # ]
    # direct = [
    #     s.split()[5] for s in inp if len(s.split()) > 4 and s.split()[2] == "Trialinfo:"
    # ]
    # Check if any actual data recorded in file
    starts = [i for i, x in enumerate(inp_first) if x == "START"]
    if not starts:
        raise ValueError("No samples or events found in .asc file.")

    # Read metadata from file before processing
    is_raw = [bool(re.match("^[0-9]", line)) for line in inp_first]

    info = get_info(
        [line for line, raw in zip(inp, is_raw) if not raw],
        [first for first, raw in zip(inp_first, is_raw) if not raw],
    )

    # Do some extra processing/sanitizing if there's HTARG info in the file
    if info["htarg"]:
        inp, info = handle_htarg(inp, info, is_raw)
```

```
    # Find blocks and mark lines between block ENDs and next block STARTs
    dividers = starts + [len(inp)]
    block = np.cumsum([x == "START" for x in inp_first])
    block = block.astype(float)

    for i in range(1, len(dividers)):
        start = dividers[i - 1]
        end = dividers[i]
        endline = [j for j, x in enumerate(inp_first[start:end]) if x == "END"]
        if endline and endline[-1] < end - start:
            block[endline[0] + start : end] += 0.5

    # Unless parsing all input, drop any lines not within a block
    block[: dividers[0] + 1] += 0.5
    if not parse_all:
        in_block = np.floor(block) == block
        inp = [line for line, block_match in zip(inp, in_block) if block_match]
        inp_first = [
            first for first, block_match in zip(inp_first, in_block) if block_match
        ]
        is_raw = [raw for raw, block_match in zip(is_raw, in_block) if block_match]
        block = block[in_block]

    block = np.array(block)

    # Initialize dictionary of data output and process different data types
    out = {}
    if samples:
        out["raw"] = process_raw(
            [line for line, raw in zip(inp, is_raw) if raw], block[is_raw], info
        )
    if events:
        is_sacc = np.array(inp_first) == "ESACC"
        out["sacc"] = process_saccades(
            np.array(inp)[is_sacc], np.array(block)[is_sacc], info
        )

        is_fix = np.array(inp_first) == "EFIX"
        out["fix"] = process_fixations(
            np.array(inp)[is_fix], np.array(block)[is_fix], info
        )

        is_blink = np.array(inp_first) == "EBLINK"
        out["blinks"] = process_blinks(
            np.array(inp)[is_blink], np.array(block)[is_blink]
```

```
        )

        is_msg = np.array(inp_first) == "MSG"
        out["msg"] = process_messages(np.array(inp)[is_msg], np.
↪array(block)[is_msg])

        is_input = np.array(inp_first) == "INPUT"
        out["input"] = process_input(np.array(inp)[is_input], np.
↪array(block)[is_input])

        is_button = np.array(inp_first) == "BUTTON"
        out["button"] = process_buttons(
            np.array(inp)[is_button], np.array(block)[is_button]
        )

    # needed for parsing, but otherwise redundant with CR
    info["tracking"] = None

    out["info"] = info

    return out


# event_path = "./ColorCue/data/sub-01/sub-01_col50-dir25_events.tsv"
# events = pd.read_csv(event_path, sep="\t")
# events.head()
# events
# data['info']
#
# MSG=data["msg"]
# Zero=MSG.loc[MSG.text=='TargetOn',["trial","time"]]
#
# Zero
# MSG.text.unique()
```

[ ]:
```
def process_data_file(f):
    # Read data from file
    data = read_asc(f)

    # Extract relevant data from the DataFrame
    df = data["raw"]
    mono = data["info"]["mono"]

    # Convert columns to numeric
    numeric_columns = ["trial", "time", "input"]
    if not mono:
        numeric_columns.extend(["xpl", "ypl", "psl", "xpr", "ypr", "psr"])
```

```
    else:
        numeric_columns.extend(["xp", "yp", "ps"])

    df[numeric_columns] = df[numeric_columns].apply(pd.to_numeric,␣
↪errors="coerce")

    # Drop rows where trial is equal to 1
    df = df[df["trial"] != 1]

    # Decrement the values in the 'trial' column by 1
    df.loc[:, "trial"] = df["trial"] - 1

    # Reset index after dropping rows and modifying the 'trial' column
    # df = df.reset_index(drop=True)

    # Extract messages from eyelink
    MSG = data["msg"]
    tON = MSG.loc[MSG.text == "FixOn", ["trial", "time"]]
    t0 = MSG.loc[MSG.text == "FixOff", ["trial", "time"]]
    Zero = MSG.loc[MSG.text == "TargetOn", ["trial", "time"]]

    # Reset time based on 'Zero' time
    for i in range(len(Zero)):
        df.loc[df["trial"] == i + 1, "time"] = (
            df.loc[df["trial"] == i + 1, "time"] - Zero.time.values[i]
        )

    # Drop bad trials
    # badTrials = get_bad_trials(df)
    # df = drop_bad_trials(df, badTrials)
    # Zero = drop_bad_trials(Zero, badTrials)
    # tON = drop_bad_trials(tON, badTrials)
    # t0 = drop_bad_trials(t0, badTrials)

    # common_trials = Zero["trial"].values
    # t0 = t0[t0["trial"].isin(common_trials)]
    # tON = tON[tON["trial"].isin(common_trials)]

    SON = tON.time.values - Zero.time.values
    SOFF = t0.time.values - Zero.time.values
    # ZEROS = Zero.time.values

    # Extract saccades data
    Sacc = data["sacc"]

    # Drop rows where trial is equal to 1
    Sacc = Sacc[Sacc["trial"] != 1]
```

```
# Decrement the values in the 'trial' column by 1
Sacc.loc[:, "trial"] = Sacc["trial"] - 1

# Reset saccade times
for t in Zero.trial:
    Sacc.loc[Sacc.trial == t, ["stime", "etime"]] = (
        Sacc.loc[Sacc.trial == t, ["stime", "etime"]].values
        - Zero.loc[Zero.trial == t, "time"].values
    )

# Sacc = drop_bad_trials(Sacc, badTrials)

# Extract trials with saccades within the time window [0, 80ms]
trialSacc = Sacc[(Sacc.stime >= -200) & (Sacc.etime < 80) & (Sacc.eye ==␣
↪"R")][
    "trial"
].values

saccDir = np.sign(
    (
        Sacc[(Sacc.stime >= -200) & (Sacc.etime < 80) & (Sacc.eye == "R")].
↪exp
        - Sacc[(Sacc.stime >= -200) & (Sacc.etime < 80) & (Sacc.eye ==␣
↪"R")].sxp
    ).values
)

for t in Sacc.trial.unique():
    start = Sacc.loc[(Sacc.trial == t) & (Sacc.eye == "R"), "stime"]
    end = Sacc.loc[(Sacc.trial == t) & (Sacc.eye == "R"), "etime"]

    for i in range(len(start)):
        if not mono:
            df.loc[
                (df.trial == t)
                & (df.time >= start.iloc[i] - 20)
                & (df.time <= end.iloc[i] + 20),
                "xpr",
            ] = np.nan
        else:
            df.loc[
                (df.trial == t)
                & (df.time >= start.iloc[i] - 20)
                & (df.time <= end.iloc[i] + 20),
                "xp",
            ] = np.nan
```

```python
    # Extract first bias
    # first_bias = np.where(bias == 1)[0][0]

    # Extract position and velocity data
    selected_values = (
        df.xpr[(df.time >= 80) & (df.time <= 120)]
        if not mono
        else df.xp[(df.time >= 80) & (df.time <= 120)]
    )
    posSteadyState = (
        df.xpr[(df.time >= 300) & (df.time <= 340)]
        if not mono
        else df.xp[(df.time >= 300) & (df.time <= 340)]
    )
    veloSteadyState = np.gradient(posSteadyState.values) * 1000 / 30
    # Rescale position
    pos_before = (
        df.xpr[(df.time >= -40) & (df.time <= 0)]
        if not mono
        else df.xp[(df.time >= -40) & (df.time <= 0)]
    )

    time_dim = 41
    trial_dim = len(selected_values) // time_dim

    pos = np.array(selected_values[: time_dim * trial_dim]).reshape(trial_dim,␣
↪time_dim)
    stdPos = np.std(pos, axis=1) / 30

    pos_before_reshaped = np.array(pos_before[: time_dim * trial_dim]).reshape(
        trial_dim, time_dim
    )
    pos_before_mean = np.nanmean(pos_before_reshaped, axis=1)
    # Reshaping veloSteadyState
    veloSteadyState = np.array(veloSteadyState[: trial_dim * time_dim]).reshape(
        trial_dim, time_dim
    )
    velo = np.gradient(pos, axis=1) * 1000 / 30
    velo[(velo > 20) | (velo < -20)] = np.nan

    for i, pp in enumerate(pos_before_mean):
        if pd.notna(pp):
            pos[i] = (pos[i] - pp) / 30

    # pos[(pos > 3) | (pos < -3)] = np.nan
```

```
    meanPos = np.nanmean(pos, axis=1)
    meanVelo = np.nanmean(velo, axis=1)
    stdVelo = np.std(velo, axis=1)
    meanVSS = np.nanmean(veloSteadyState, axis=1)
    # TS = trialSacc
    # SaccD = saccDir
    # SACC = Sacc

    return pd.DataFrame(
        {
            "SON": SON,
            "SOFF": SOFF,
            "meanPos": meanPos,
            "stdPos": stdPos,
            "meanVelo": meanVelo,
            "stdVelo": stdVelo,
            "meanVSS": meanVSS,
            # "TS": TS,
            # "SaccD": SaccD,
            # "SACC": SACC
        }
    )
```

[ ]:
```
def process_all_asc_files(data_dir):
    allDFs = []
    allEvents = []

    for root, _, files in sorted(os.walk(data_dir)):
        for filename in sorted(files):
            if filename.endswith(".asc"):
                filepath = os.path.join(root, filename)
                print(f"Read data from {filepath}")
                data = process_data_file(filepath)
                # Extract proba from filename
                proba = int(re.search(r"dir(\d+)", filename).group(1))
                data["proba"] = proba

                allDFs.append(data)
                print(len(data))

            if filename.endswith(".tsv"):
                filepath = os.path.join(root, filename)
                print(f"Read data from {filepath}")
                events = pd.read_csv(filepath, sep="\t")
                # Extract proba from filename
                # proba = int(re.search(r"dir(\d+)", filename).group(1))
                # events['proba'] = proba
```

```
                # print(len(events))
                allEvents.append(events)

    bigDF = pd.concat(allDFs, axis=0, ignore_index=True)
    # print(len(bigDF))
    bigEvents = pd.concat(allEvents, axis=0, ignore_index=True)
    # print(len(bigEvents))
    # Merge DataFrames based on 'proba'
    merged_data = pd.concat([bigEvents, bigDF], axis=1)
    # print(len(merged_data))

    return merged_data
```

[ ]:
```
path = "/Volumes/work/brainets/oueld.h/contextuaLearning/data/"
```

[1]:
```
df = process_all_asc_files(path)
```

```
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-01/sub-
01_col50-dir25_events.tsv
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-01/sub-
01_col50-dir25_eyeData.asc
240
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-01/sub-
01_col50-dir50_events.tsv
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-01/sub-
01_col50-dir50_eyeData.asc
240
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-01/sub-
01_col50-dir75_events.tsv
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-01/sub-
01_col50-dir75_eyeData.asc
240
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-02/sub-
02_col50-dir25_events.tsv
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-02/sub-
02_col50-dir25_eyeData.asc
240
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-02/sub-
02_col50-dir50_events.tsv
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-02/sub-
02_col50-dir50_eyeData.asc
240
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-02/sub-
02_col50-dir75_events.tsv
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-02/sub-
02_col50-dir75_eyeData.asc
240
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-03/sub-
```

```
03_col50-dir25_events.tsv
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-03/sub-
03_col50-dir25_eyeData.asc
240
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-03/sub-
03_col50-dir50_events.tsv
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-03/sub-
03_col50-dir50_eyeData.asc
240
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-03/sub-
03_col50-dir75_events.tsv
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-03/sub-
03_col50-dir75_eyeData.asc
240
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-04/sub-
04_col50-dir25_events.tsv
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-04/sub-
04_col50-dir25_eyeData.asc
240
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-04/sub-
04_col50-dir50_events.tsv
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-04/sub-
04_col50-dir50_eyeData.asc
240
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-04/sub-
04_col50-dir75_events.tsv
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-04/sub-
04_col50-dir75_eyeData.asc
240
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-05/sub-
05_col50-dir25_events.tsv
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-05/sub-
05_col50-dir25_eyeData.asc
240
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-05/sub-
05_col50-dir50_events.tsv
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-05/sub-
05_col50-dir50_eyeData.asc
240
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-05/sub-
05_col50-dir75_events.tsv
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-05/sub-
05_col50-dir75_eyeData.asc
240
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-06/sub-
06_col50-dir25_events.tsv
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-06/sub-
06_col50-dir25_eyeData.asc
```

```
240
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-06/sub-
06_col50-dir50_events.tsv
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-06/sub-
06_col50-dir50_eyeData.asc
240
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-06/sub-
06_col50-dir75_events.tsv
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-06/sub-
06_col50-dir75_eyeData.asc
240
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-07/sub-
07_col50-dir25_events.tsv
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-07/sub-
07_col50-dir25_eyeData.asc
240
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-07/sub-
07_col50-dir50_events.tsv
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-07/sub-
07_col50-dir50_eyeData.asc
240
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-07/sub-
07_col50-dir75_events.tsv
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-07/sub-
07_col50-dir75_eyeData.asc
240
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-08/sub-
08_col50-dir25_events.tsv
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-08/sub-
08_col50-dir25_eyeData.asc
240
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-08/sub-
08_col50-dir50_events.tsv
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-08/sub-
08_col50-dir50_eyeData.asc
240
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-08/sub-
08_col50-dir75_events.tsv
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-08/sub-
08_col50-dir75_eyeData.asc
240
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-09/sub-
09_col50-dir25_events.tsv
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-09/sub-
09_col50-dir25_eyeData.asc
240
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-09/sub-
09_col50-dir50_events.tsv
```

```
Read data from /Volumes/work/brainets/oueld.h/contextualLearning/data/sub-09/sub-
09_col50-dir50_eyeData.asc
240
Read data from /Volumes/work/brainets/oueld.h/contextualLearning/data/sub-09/sub-
09_col50-dir75_events.tsv
Read data from /Volumes/work/brainets/oueld.h/contextualLearning/data/sub-09/sub-
09_col50-dir75_eyeData.asc
240
Read data from /Volumes/work/brainets/oueld.h/contextualLearning/data/sub-10/sub-
10_col50-dir25_events.tsv
Read data from /Volumes/work/brainets/oueld.h/contextualLearning/data/sub-10/sub-
10_col50-dir25_eyeData.asc
240
Read data from /Volumes/work/brainets/oueld.h/contextualLearning/data/sub-10/sub-
10_col50-dir50_events.tsv
Read data from /Volumes/work/brainets/oueld.h/contextualLearning/data/sub-10/sub-
10_col50-dir50_eyeData.asc
240
Read data from /Volumes/work/brainets/oueld.h/contextualLearning/data/sub-10/sub-
10_col50-dir75_events.tsv
Read data from /Volumes/work/brainets/oueld.h/contextualLearning/data/sub-10/sub-
10_col50-dir75_eyeData.asc
240
Read data from /Volumes/work/brainets/oueld.h/contextualLearning/data/sub-11/sub-
11_col50-dir25_events.tsv
Read data from /Volumes/work/brainets/oueld.h/contextualLearning/data/sub-11/sub-
11_col50-dir25_eyeData.asc
240
Read data from /Volumes/work/brainets/oueld.h/contextualLearning/data/sub-11/sub-
11_col50-dir50_events.tsv
Read data from /Volumes/work/brainets/oueld.h/contextualLearning/data/sub-11/sub-
11_col50-dir50_eyeData.asc
240
Read data from /Volumes/work/brainets/oueld.h/contextualLearning/data/sub-11/sub-
11_col50-dir75_events.tsv
Read data from /Volumes/work/brainets/oueld.h/contextualLearning/data/sub-11/sub-
11_col50-dir75_eyeData.asc
240
Read data from /Volumes/work/brainets/oueld.h/contextualLearning/data/sub-12/sub-
12_col50-dir25_events.tsv
Read data from /Volumes/work/brainets/oueld.h/contextualLearning/data/sub-12/sub-
12_col50-dir25_eyeData.asc
240
Read data from /Volumes/work/brainets/oueld.h/contextualLearning/data/sub-12/sub-
12_col50-dir50_events.tsv
Read data from /Volumes/work/brainets/oueld.h/contextualLearning/data/sub-12/sub-
12_col50-dir50_eyeData.asc
240
```

```
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-12/sub-
12_col50-dir75_events.tsv
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-12/sub-
12_col50-dir75_eyeData.asc
240
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-13/sub-
13_col50-dir25_events.tsv
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-13/sub-
13_col50-dir25_eyeData.asc
240
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-13/sub-
13_col50-dir50_events.tsv
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-13/sub-
13_col50-dir50_eyeData.asc
240
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-13/sub-
13_col50-dir75_events.tsv
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-13/sub-
13_col50-dir75_eyeData.asc
240
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-14/sub-
14_col50-dir25_events.tsv
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-14/sub-
14_col50-dir25_eyeData.asc
240
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-14/sub-
14_col50-dir50_events.tsv
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-14/sub-
14_col50-dir50_eyeData.asc
240
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-14/sub-
14_col50-dir75_events.tsv
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-14/sub-
14_col50-dir75_eyeData.asc
240
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-15/sub-
15_col50-dir25_events.tsv
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-15/sub-
15_col50-dir25_eyeData.asc
240
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-15/sub-
15_col50-dir50_events.tsv
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-15/sub-
15_col50-dir50_eyeData.asc
240
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-15/sub-
15_col50-dir75_events.tsv
Read data from /Volumes/work/brainets/oueld.h/contextuaLearning/data/sub-15/sub-
```

```
15_col50-dir75_eyeData.asc
240
Read data from /Volumes/work/brainets/oueld.h/contextualLearning/data/sub-16/sub-
16_col50-dir25_events.tsv
Read data from /Volumes/work/brainets/oueld.h/contextualLearning/data/sub-16/sub-
16_col50-dir25_eyeData.asc
240
Read data from /Volumes/work/brainets/oueld.h/contextualLearning/data/sub-16/sub-
16_col50-dir50_events.tsv
Read data from /Volumes/work/brainets/oueld.h/contextualLearning/data/sub-16/sub-
16_col50-dir50_eyeData.asc
240
Read data from /Volumes/work/brainets/oueld.h/contextualLearning/data/sub-16/sub-
16_col50-dir75_events.tsv
Read data from /Volumes/work/brainets/oueld.h/contextualLearning/data/sub-16/sub-
16_col50-dir75_eyeData.asc
240
```

[2]:
```
df.head()
```

[2]:
```
         onset  duration  sub_number  …   stdVelo    meanVSS  proba
0  8791.399394  1.458303           1  …  3.746900   2.113821     25
1  8794.490970  1.449872           1  …       NaN  -0.894309     25
2  8797.507458  1.508185           1  …  4.726648 -56.504065     25
3  8800.365510  1.333316           1  …  4.322727 -71.138211     25
4  8803.198679  1.416712           1  …  3.186847  -4.837398     25

[5 rows x 16 columns]
```

[3]:
```
df.meanVelo.isna().sum()
```

[3]:
```
22
```

[4]:
```
data=df.copy()
df.to_csv("data.csv", index=False)
data
```

[4]:
```
             onset  duration  sub_number  …   stdVelo    meanVSS  proba
0      8791.399394  1.458303           1  …  3.746900   2.113821     25
1      8794.490970  1.449872           1  …       NaN  -0.894309     25
2      8797.507458  1.508185           1  …  4.726648 -56.504065     25
3      8800.365510  1.333316           1  …  4.322727 -71.138211     25
4      8803.198679  1.416712           1  …  3.186847  -4.837398     25
```

```
...             ...         ...           ...  ...        ...         ...     ...
11515  2905.713837  1.458243           16  ...  6.330771  76.178862     75
11516  2908.605220  1.358295           16  ...  4.493663 -54.796748     75
11517  2911.371848  1.458237           16  ...  6.593140 -67.642276     75
11518  2914.271578  1.541621           16  ...  4.131169  35.813008     75
11519  2917.279748  1.458303           16  ...  4.047115  60.975610     75

[11520 rows x 16 columns]
```

[ ]:
```
data[(data.sub_number==8) & (data.proba==75)]
```

1 Start Running the code from Here

[5]:
```
df = pd.read_csv("data.csv")
# [print(df[df["sub_number"] == i]["meanVelo"].isna().sum()) for i in range(1,␣
↪13)]
# df.dropna(inplace=True)
df["color"] = df["trial_color_chosen"].apply(lambda x: "green" if x == 0 else␣
↪"red")


df = df.dropna(subset=['meanVelo'])
# Assuming your DataFrame is named 'df' and the column you want to rename is␣
↪'old_column'
# df.rename(columns={'old_column': 'new_column'}, inplace=True)
df.head()
```

[5]:
```
         onset  duration  sub_number  trial_number  ...   stdVelo    meanVSS
proba  color
0  8791.399394  1.458303           1             1  ...  3.746900   2.113821
25  green
1  8794.490970  1.449872           1             2  ...       NaN  -0.894309
25    red
2  8797.507458  1.508185           1             3  ...  4.726648 -56.504065
25  green
3  8800.365510  1.333316           1             4  ...  4.322727 -71.138211
25    red
4  8803.198679  1.416712           1             5  ...  3.186847  -4.837398
25    red

[5 rows x 17 columns]
```

[6]:
```
df.meanVelo.isna().sum()
```

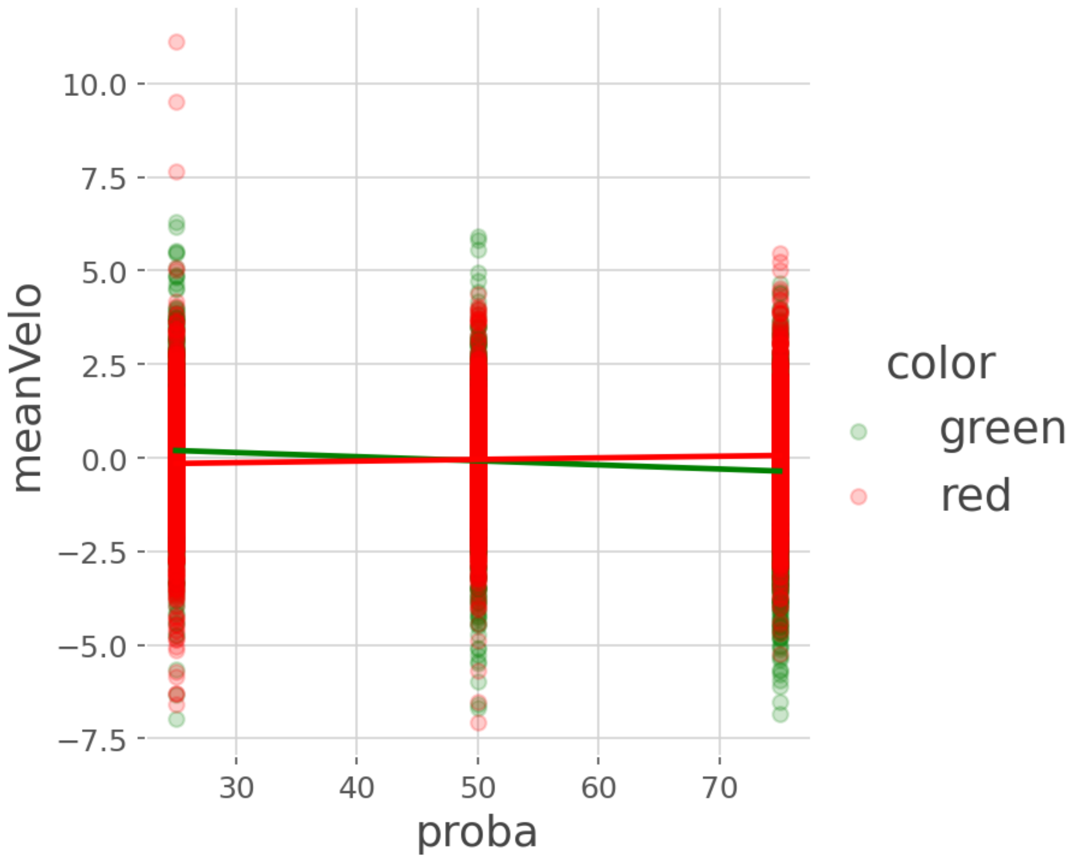
[6]: 0

[ ]:
```
df = df[df['sub_number'] != 9]
```

[ ]:
```
colors = ["green", "red"]
# Set style to whitegrid

# # Set font size for labels
# sns.set(
#     rc={
#         "axes.labelsize": 25,
#
#         "axes.titlesize": 20,
#     }
# )
#
# sns.set_style("whitegrid")
```

[7]:
```
sns.lmplot(
    x="proba",
    y="meanVelo",
    data=df,
    hue="color",
    scatter_kws={"alpha": 0.2},
    palette=colors,
)
```

```
Installed osx event loop hook.
```



[7]: <seaborn.axisgrid.FacetGrid at 0x10d47dc90>

[8]:
```
l = (
    df.groupby(["sub_number", "trial_color_chosen", "proba"])
    .meanVelo.mean()
    .reset_index())

l
```

[8]:

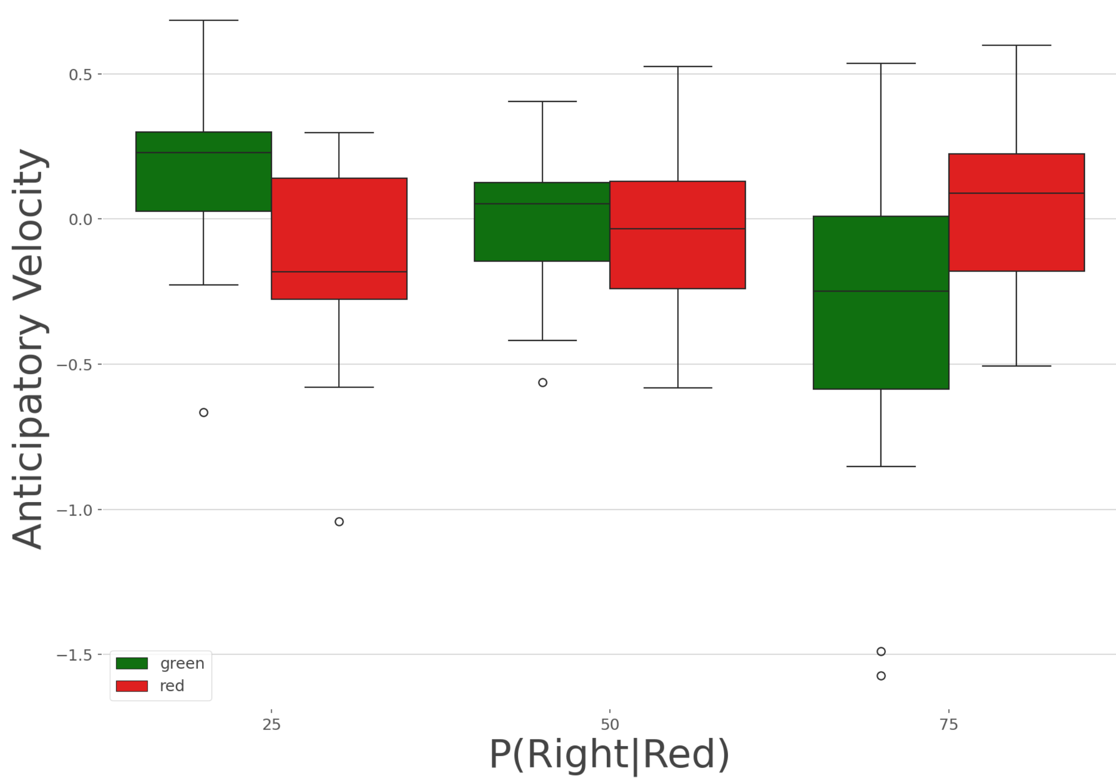
	sub_number	trial_color_chosen	proba	meanVelo
0	1	0	25	-0.004063
1	1	0	50	-0.088449
2	1	0	75	-0.851774
3	1	1	25	-0.281891
4	1	1	50	-0.052357
..	…	…	…	…
85	16	0	50	0.072125

```
86          16                    0     75 -0.194897
87          16                    1     25  0.150033
88          16                    1     50 -0.033862
89          16                    1     75 -0.349932

[90 rows x 4 columns]
```

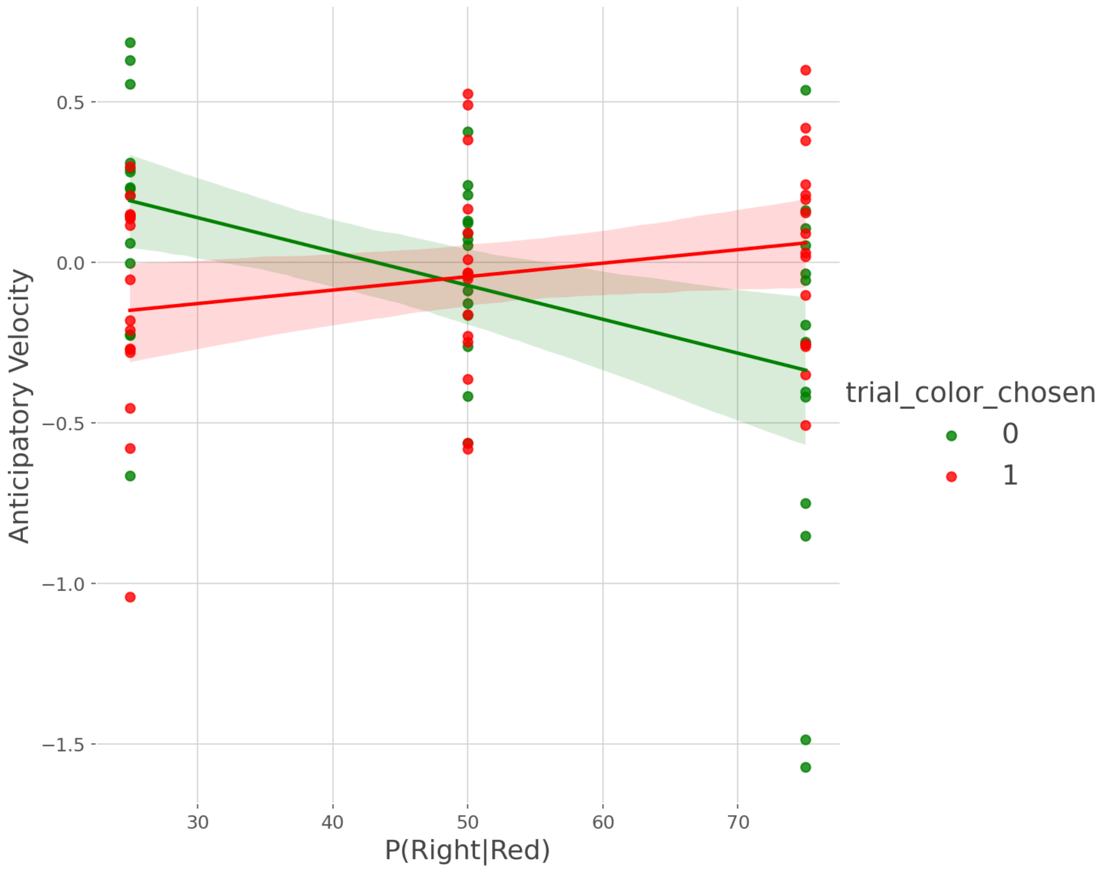
[ ]:
```
l["color"] = l["trial_color_chosen"].apply(lambda x: "green" if x == 0 else␣
↪"red")
```

[9]:
```
bp = sns.boxplot(
    x="proba", y="meanVelo", hue="color", data=l, palette=colors
)
bp.legend(fontsize="larger")
plt.xlabel("P(Right|Red)", fontsize=30)
plt.ylabel("Anticipatory Velocity", fontsize=30)
plt.savefig("clccbp.png")
```

```
Installed osx event loop hook.
```



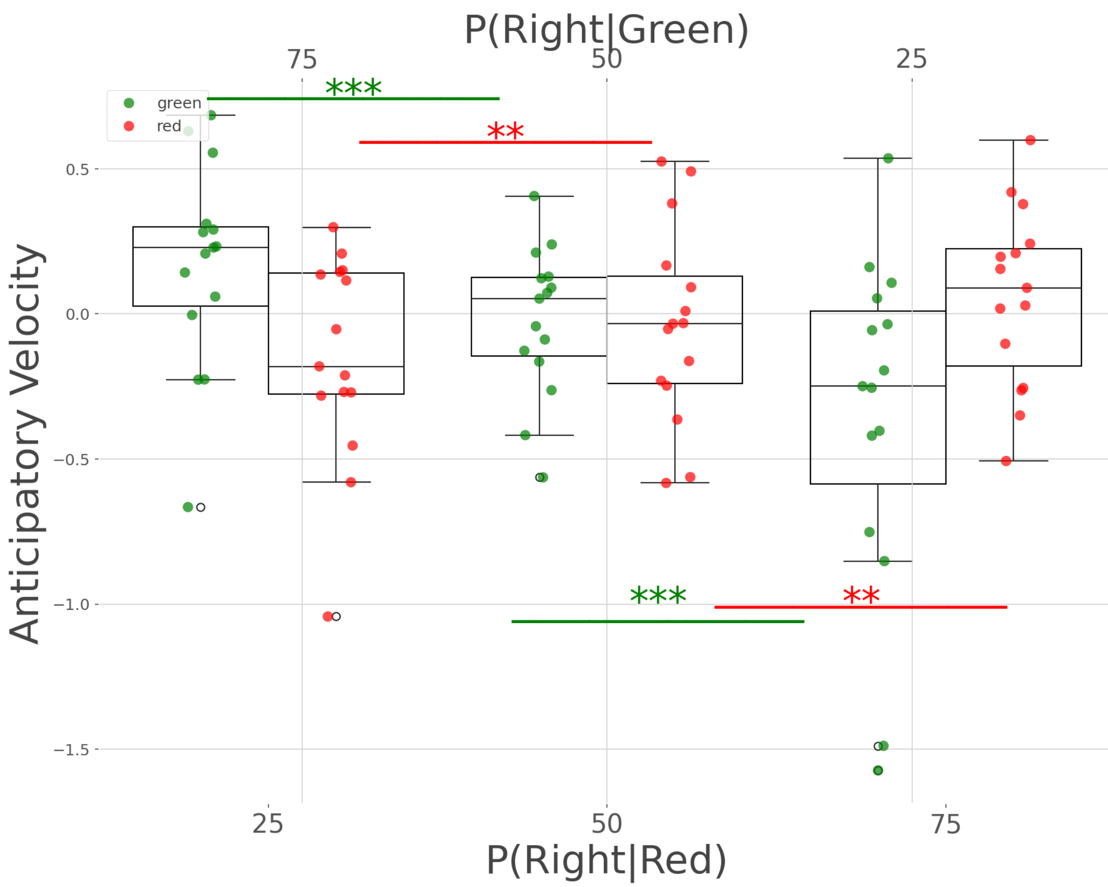
[10]:
```
lm = sns.lmplot(
    x="proba", y="meanVelo", hue="trial_color_chosen", data=l, palette=colors,␣
↪height=8
)
# Adjust font size for axis labels
lm.set_axis_labels("P(Right|Red)", "Anticipatory Velocity")
# lm.ax.legend(fontsize='large')
plt.savefig("clcclp.png")
```



[11]:
```
# Create the box plot with transparent fill and black borders, and without␣
↪legend
bp = sns.boxplot(
    x="proba", y="meanVelo", hue="color", data=l, palette=colors,
    boxprops=dict(facecolor='none', edgecolor='black'), legend=False
)

# Add scatter plot on top
sns.stripplot(
```

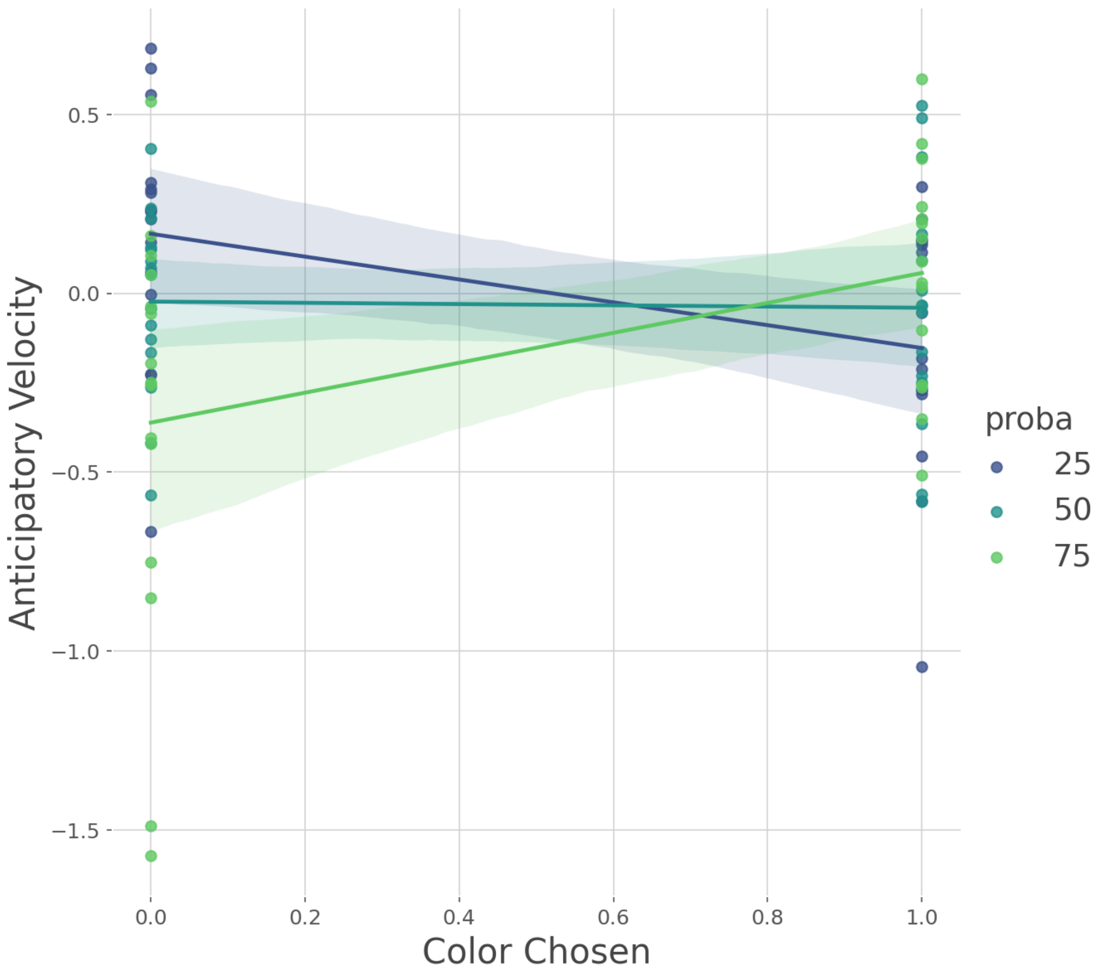
```
    x="proba", y="meanVelo", hue="color", data=l, dodge=True, palette=colors,␣
 ↪jitter=True, size=8, alpha=0.7
)
# Set labels for both top and bottom x-axes
plt.xlabel("P(Right|Red)", fontsize=30)
plt.ylabel("Anticipatory Velocity", fontsize=30)
plt.xticks( fontsize=20)

#Overlay regplot on top of the boxplot and stripplot

plt.twiny().set_xlabel("P(Right|Green)", fontsize=30)
# Set the tick positions for both top and bottom x-axes
tick_positions = [0.2, 0.5, 0.8]
tick_labels = [25, 50, 75]

# Set the ticks and labels for both top and bottom x-axes
plt.xticks(tick_positions, tick_labels, fontsize=20)
plt.xticks(fontsize=20)
# Invert the top x-axis
plt.gca().invert_xaxis()

# Manually add stars indicating statistical significance
# Adjust the coordinates based on your plot
plt.text(0.6, 0.6, '**', fontsize=30, ha='center', va='center', color='red')
plt.text(0.6, 0.65, '_______________', fontsize=30, ha='center', va='center',␣
 ↪color='red')
# plt.text(0.6, 0.6, 'p < 0.001', fontsize=15, ha='center', va='center',␣
 ↪color='red')

plt.text(0.75, 0.75, '***', fontsize=30, ha='center', va='center',␣
 ↪color='green')
plt.text(0.75, 0.8, '_______________', fontsize=30, ha='center', va='center',␣
 ↪color='green')

# Right side

plt.text(0.25,-1, '**', fontsize=30, ha='center', va='center', color='red')
plt.text(0.25,- 0.95, '_______________', fontsize=30, ha='center', va='center',␣
 ↪color='red')
# plt.text(0.6, 0.6, 'p < 0.001', fontsize=15, ha='center', va='center',␣
 ↪color='red')

plt.text(0.45,- 1, '***', fontsize=30, ha='center', va='center', color='green')
plt.text(0.45, -1, '_______________', fontsize=30, ha='center', va='center',␣
 ↪color='green')
```

```
# plt.text(0.333, 0.6, 'p < 0.001', fontsize=15, ha='center', va='center',␣
↪color='green')
# Adjust legend
bp.legend(fontsize="larger", loc="upper left")


# Save the plot
plt.savefig("clccbp.png")

# Show the plot
plt.show()
```



[12]:
```
lm = sns.lmplot(
    x="trial_color_chosen",
    y="meanVelo",
    hue="proba",
    data=l,
```

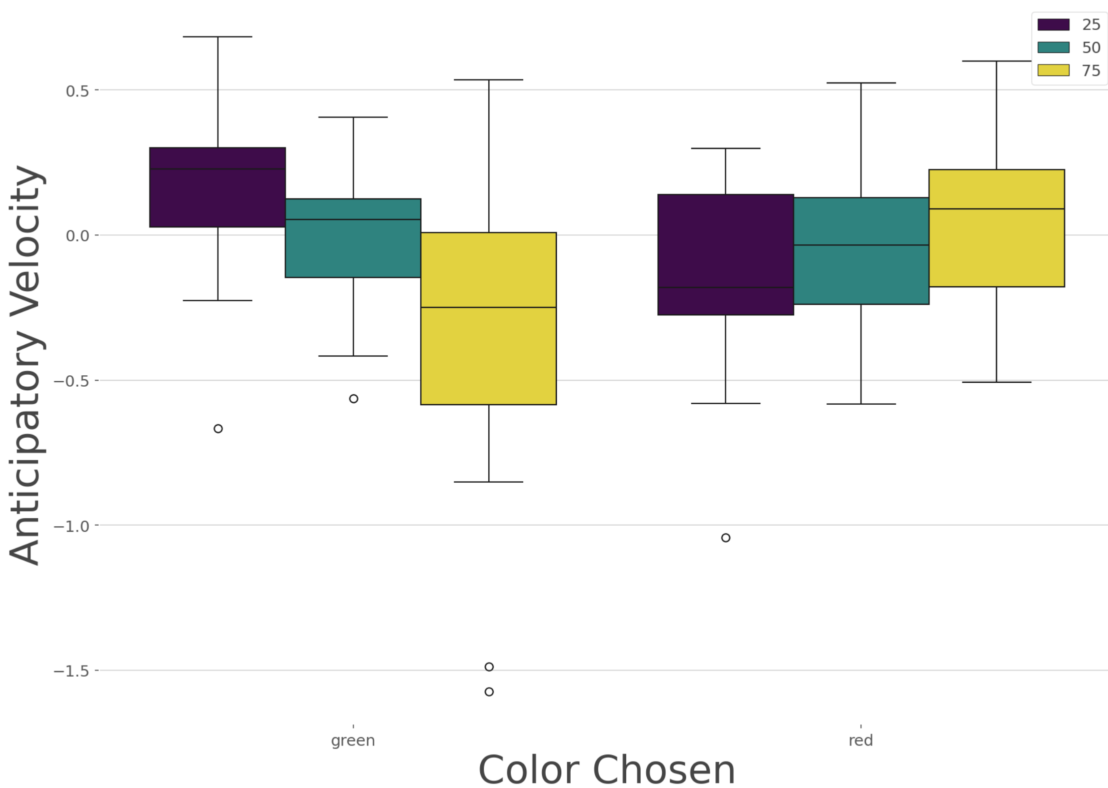
```
    height=8,
    palette="viridis",
)
# Adjust font size for axis labels
lm.set_axis_labels("Color Chosen", "Anticipatory Velocity", fontsize=20)
```



[12]: <seaborn.axisgrid.FacetGrid at 0x1506fda50>

[13]:
```
bp = sns.boxplot(
    x="color",
    y="meanVelo",
    hue="proba",
    data=l,
```

```
    palette="viridis",
)

bp.legend(fontsize="larger")
plt.xlabel("Color Chosen", fontsize=30)
plt.ylabel("Anticipatory Velocity", fontsize=30)
plt.savefig('antihueproba.png')
```



[ ]:
```
df[(df.sub_number == 8)].trial_color_chosen
```

[ ]:
```
model = sm.OLS.from_formula("meanVelo ~ C(proba) ", data=df[df.color == "red"])
result = model.fit()

print(result.summary())
```

[ ]:
```
model = sm.OLS.from_formula("meanVelo ~ C(proba) ", data=df[df.color ==␣
 ↪"green"])
result = model.fit()

print(result.summary())
```

[ ]:
```
model = sm.OLS.from_formula("meanVelo ~ C(color) ", data=df[df.proba == 25])
result = model.fit()

print(result.summary())
```

[ ]:
```
model = sm.OLS.from_formula("meanVelo ~ C(color) ", data=df[df.proba == 75])
result = model.fit()

print(result.summary())
```

[ ]:
```
model = sm.OLS.from_formula("meanVelo ~ C(color) ", data=df[df.proba == 50])
result = model.fit()

print(result.summary())
```

[ ]:
```
model = ols("meanVelo ~ C(proba) ", data=df[df.trial_color_chosen == 1]).fit()
anova_table = sm.stats.anova_lm(model, typ=3)

print(anova_table)
```

[ ]:
```
rp.summary_cont(df.groupby(["sub_number", "color", "proba"])["meanVelo"])
```

[ ]:
```
model = ols("meanVelo ~ C(proba):C(color) ", data=df).fit()
anova_table = sm.stats.anova_lm(model, typ=3)

print(anova_table)
```

[14]:
```
model = smf.mixedlm(
    "meanVelo ~ C(color)*C(proba)",
    data=df,
    groups=df["sub_number"],
).fit()
model.summary()
```

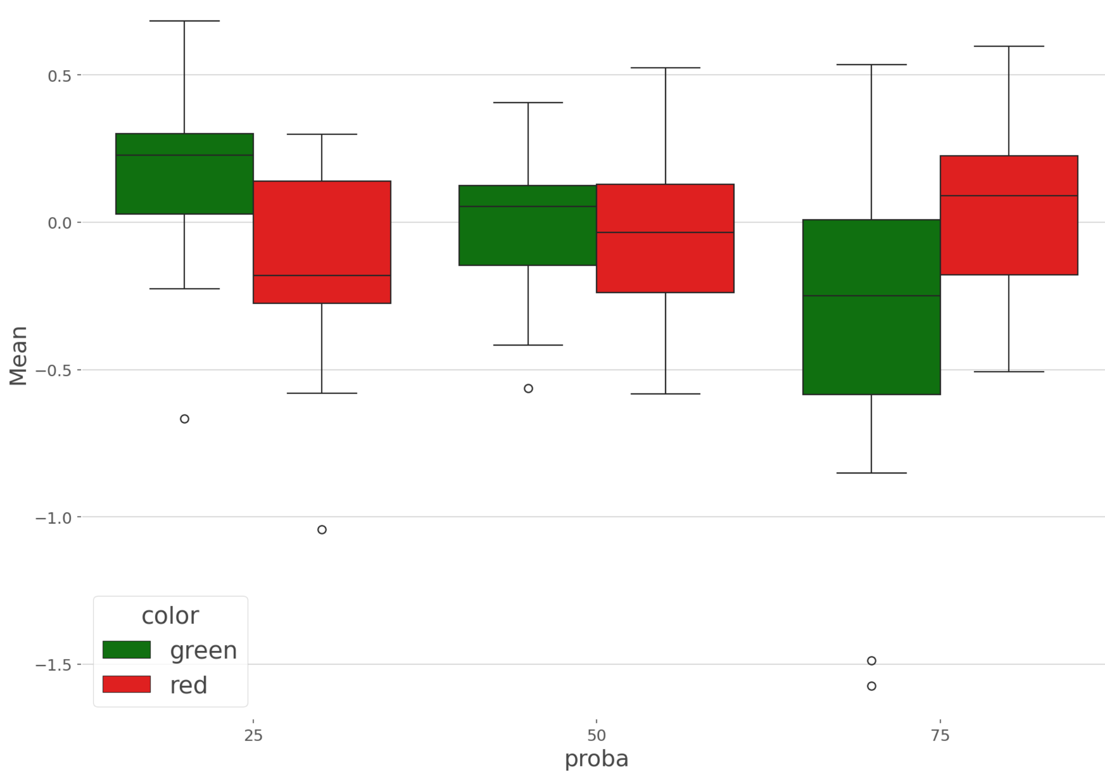
[14]:
```
<class 'statsmodels.iolib.summary2.Summary'>
"""
                  Mixed Linear Model Regression Results
==========================================================================
Model:                    MixedLM     Dependent Variable:     meanVelo
No. Observations:         10779       Method:                 REML
No. Groups:               15          Scale:                  1.8108
Min. group size:          711         Log-Likelihood:         -18532.7258
Max. group size:          720         Converged:              Yes
Mean group size:          718.6
--------------------------------------------------------------------------
                               Coef.  Std.Err.    z    P>|z| [0.025 0.975]
```

```
-----------------------------------------------------------------------
Intercept                        0.167    0.086   1.927 0.054 -0.003  0.336
C(color)[T.red]                 -0.319    0.045  -7.107 0.000 -0.407 -0.231
C(proba)[T.50]                  -0.192    0.045  -4.275 0.000 -0.279 -0.104
C(proba)[T.75]                  -0.537    0.045 -12.038 0.000 -0.625 -0.450
C(color)[T.red]:C(proba)[T.50]   0.301    0.063   4.734 0.000  0.176  0.425
C(color)[T.red]:C(proba)[T.75]   0.741    0.064  11.649 0.000  0.616  0.865
Group Var                        0.097    0.028
=======================================================================

"""
```

[15]:
```python
summary = rp.summary_cont(
    df.groupby(["sub_number", "color", "proba"])["meanVelo"]
)
```

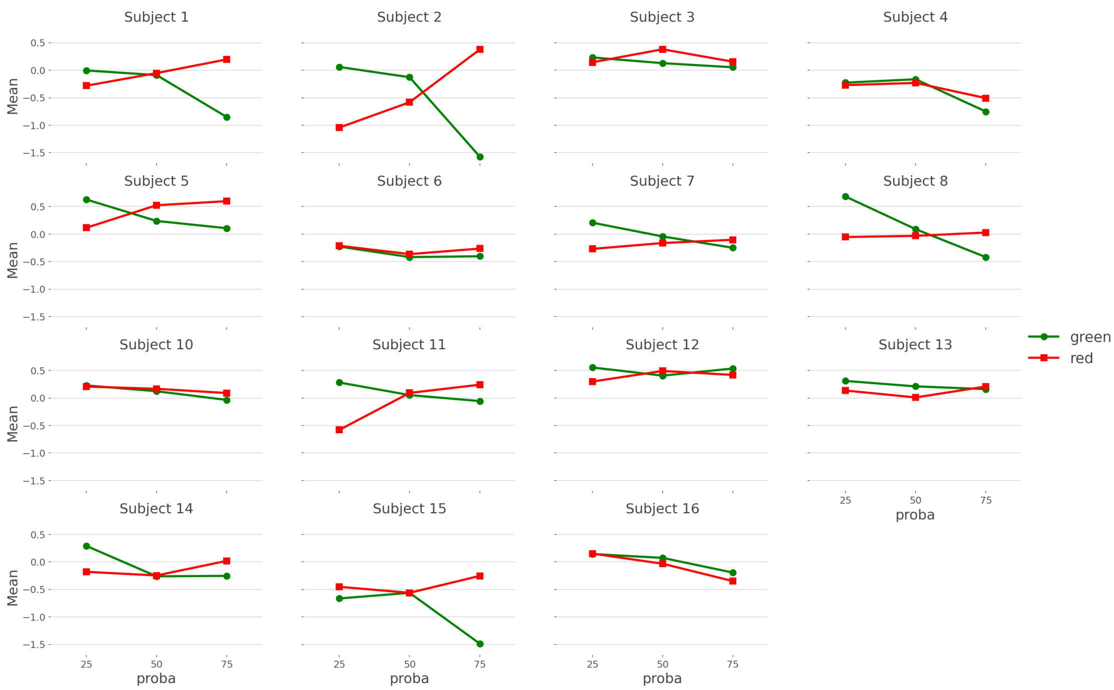
[ ]:
```python
summary.reset_index(inplace=True)
```

[16]:
```python
sns.boxplot(data=summary, x="proba", y="Mean", hue="color",palette=["green",␣
 ↪"red"])
```

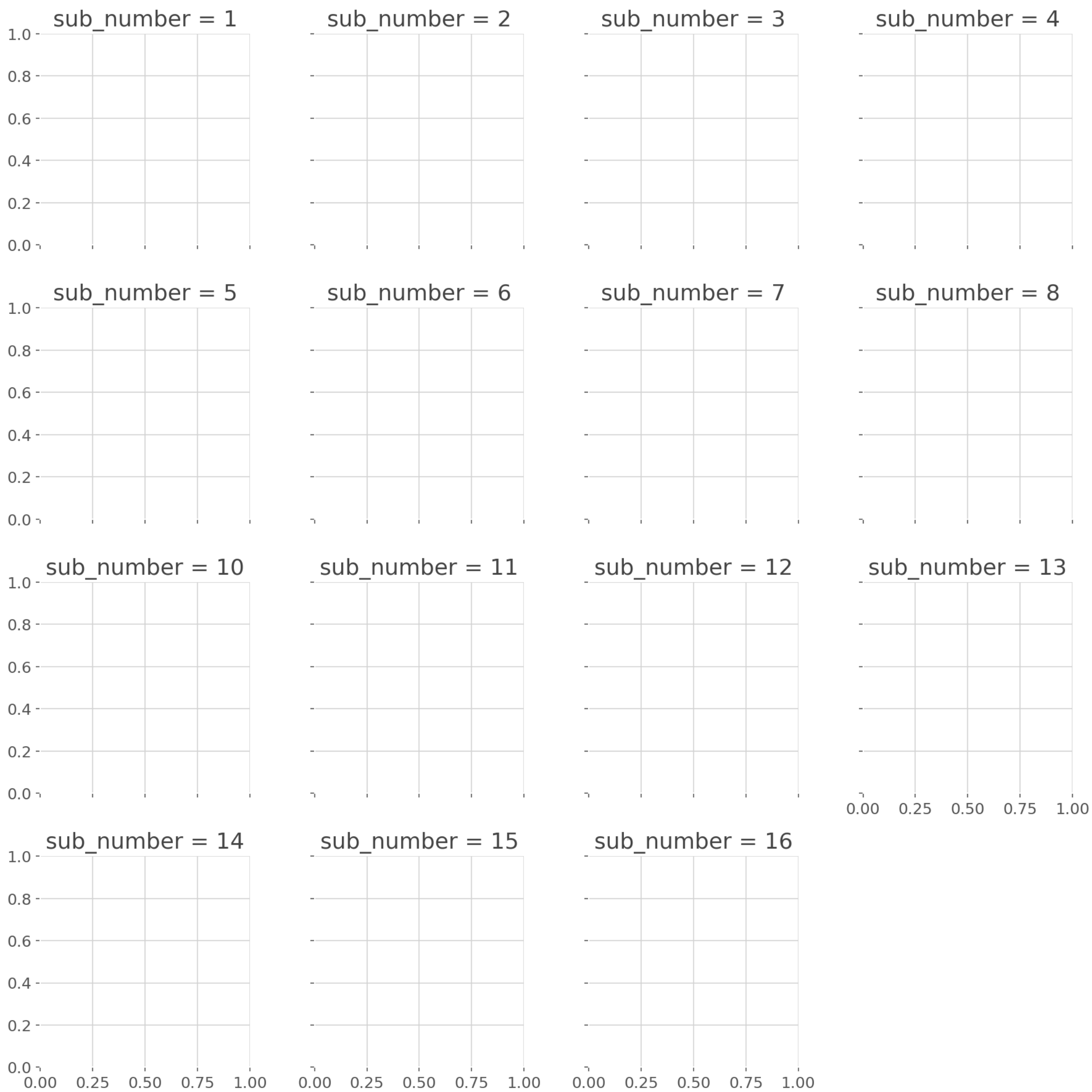


[16]: <Axes: xlabel='proba', ylabel='Mean'>

[17]:
```
# Get unique sub_numbers
unique_sub_numbers = summary["sub_number"].unique()

# Set up the FacetGrid
facet_grid = sns.FacetGrid(data=summary, col="sub_number", col_wrap=4,␣
↪sharex=True, sharey=True, height=3, aspect=1.5)

# Create pointplots for each sub_number
facet_grid.map_dataframe(sns.pointplot ,x="proba", y="Mean", hue="color",␣
↪markers=["o", "s", "d"], palette=['green','red'])

# Add legends
facet_grid.add_legend()

# Set titles for each subplot
for ax, sub_number in zip(facet_grid.axes.flat, unique_sub_numbers):
    ax.set_title(f"Subject {sub_number}")

# Adjust spacing between subplots
facet_grid.fig.subplots_adjust(wspace=0.2, hspace=0.2)  # Adjust wspace and␣
↪hspace as needed

# Show the plot
plt.savefig("allSubjectanti.png")
```



[18]:
```
grid= sns.FacetGrid(df, col="sub_number",hue='proba', col_wrap=4, height=3)
```




```
[ ]: grid.map(plt.scatter,'trial_number','meanVelo')
```

```
[ ]: # Create a KDE plot of residuals
     sns.displot(model.resid, kind="kde", fill=True, lw=1)

     # Overlay normal distribution on the same plot
     xmin, xmax = plt.xlim()
     x = np.linspace(xmin, xmax, 1000)
     p = stats.norm.pdf(x, np.mean(model.resid), np.std(model.resid))
     plt.plot(x, p, "k", linewidth=1)
```

```python
# Set title and labels
plt.title("KDE Plot of Model Residuals (Red) and Normal Distribution (Black)")
plt.xlabel("Residuals")
```

[ ]:
```python
fig = plt.figure(figsize=(16, 9))
ax = fig.add_subplot(111)

sm.qqplot(model.resid, dist=stats.norm, line="s", ax=ax)

ax.set_title("Q-Q Plot")
```

[ ]:
```python
labels = ["Statistic", "p-value"]

norm_res = stats.shapiro(model.resid)

for key, val in dict(zip(labels, norm_res)).items():
    print(key, val)
```

[ ]:
```python
fig = plt.figure(figsize=(16, 9))

ax = sns.boxplot(x=model.model.groups, y=model.resid)

ax.set_title("Distribution of Residuals for Anticipatory Velocity by Subject")
ax.set_ylabel("Residuals")
ax.set_xlabel("Subject")
```

[ ]:
```python
het_white_res = het_white(model.resid, model.model.exog)

labels = ["LM Statistic", "LM-Test p-value", "F-Statistic", "F-Test p-value"]

for key, val in dict(zip(labels, het_white_res)).items():
    print(key, val)
```

[ ]:
```python
# t test to comprare proba 25/red and proba75/green
stats.ttest_ind(
    df[(df.proba == 25) & (df.color == "red")].meanVelo,
    df[(df.proba == 75) & (df.color == "green")].meanVelo,
)
```

[ ]:
```python
# t test to comprare proba 25/red and proba75/green
stats.ttest_ind(
    df[(df.proba == 75) & (df.color == "red")].meanVelo,
    df[(df.proba == 25) & (df.color == "green")].meanVelo,
)
```

[ ]:
```python
stats.ttest_ind(
    df[(df.proba == 50) & (df.color == "red")].meanVelo,
```

```
    df[(df.proba == 50) & (df.color == "green")].meanVelo,
)
```

[ ]:
```
stats.ttest_ind(
    df[(df.proba == 50) & (df.color == "green")].meanVelo,
    df[(df.proba == 75) & (df.color == "green")].meanVelo,
)
```

[ ]:
```
# Example assuming 'proba' and 'color' are categorical variables in your␣
↪DataFrame
colors = df["color"].unique()

for color in colors:
    # Filter data for the current color
    color_data = df[df["color"] == color]

    # Group data by 'proba' and get meanVelo for each group
    grouped_data = [group["meanVelo"] for proba, group in color_data.
↪groupby("proba")]

    # Perform Kruskal-Wallis test
    statistic, p_value = kruskal(*grouped_data)

    # Print results for each color
    print(f"Color: {color}")
    print(f"Kruskal-Wallis Statistic: {statistic}")
    print(f"P-value: {p_value}")

    # Check if the result is statistically significant
    if p_value < 0.01:
        print(
            "The probabilities within this color have significantly different␣
↪distributions."
        )
    else:
        print(
            "There is not enough evidence to suggest significant differences␣
↪between probabilities within this color."
        )
    print("\n")
```

2 Analysis of subject who did Vanessa's task

[ ]:
```
df_prime = df[(df.sub_number > 12) ]
```

[ ]:
```python
l_prime = (
    df_prime.groupby(["sub_number", "trial_color_chosen", "proba"])
    .meanVelo.mean()
    .reset_index()
)
l_prime
```

[ ]:
```python
bp = sns.boxplot(
    x="proba", y="meanVelo", hue="trial_color_chosen", data=l_prime, palette=colors
)
bp.legend(fontsize="larger")
plt.xlabel("P(Dir=Right|Color=Red)", fontsize=30)
plt.ylabel("Anticipatory Velocity", fontsize=30)
```

[ ]:
```python
lm = sns.lmplot(
    x="proba", y="meanVelo", hue="trial_color_chosen", data=l_prime, palette=colors, height=8
)
# Adjust font size for axis labels
lm.set_axis_labels("P(R|Red)", "Anticipatory Velocity")
```

[ ]:
```python
# Participants balanced their choices
print(df.trial_color_chosen.value_counts())
```

[ ]:
```python
from collections import Counter
def compute_probability_distribution_tplus1_given_t(df, subject_col, condition_col, choice_col):
    # df is your DataFrame
    # subject_col is the column name for the subjects
    # condition_col is the column name for the conditions
    # choice_col is the column name for the choices

    # Create a dictionary to store probability distributions for each subject and condition group
    probability_distributions = {}

    # Iterate over unique subject-condition pairs
    for (subject, condition), group_df in df.groupby([subject_col, condition_col]):
        choices = group_df[choice_col].tolist()

        # Count occurrences of each pair (C_t, C_{t+1})
        transition_counts = Counter(zip(choices[:-1], choices[1:]))

        # Compute total counts for each choice at time t
        total_counts_t = Counter(choices[:-1])
```

```
        # Calculate the conditional probabilities
        probability_distribution = {}
        for (choice_t, choice_tplus1), count in transition_counts.items():
            probability_distribution[(choice_tplus1, choice_t)] = count /␣
 ↪total_counts_t[choice_t]

        # Store the probability distribution in the dictionary
        probability_distributions[(subject, condition)] =␣
 ↪probability_distribution

    return probability_distributions
```

[19]:
```
probability_distributions_by_group =␣
 ↪compute_probability_distribution_tplus1_given_t(df, 'sub_number', 'proba',␣
 ↪'trial_color_chosen')
probability_distributions_by_group
```

[19]:
```
{(1, 25): {(1, 0): 0.2926829268292683,
  (0, 1): 0.3017241379310345,
  (1, 1): 0.6982758620689655,
  (0, 0): 0.7073170731707317},
 (1, 50): {(0, 0): 0.6058394160583942,
  (1, 0): 0.39416058394160586,
  (1, 1): 0.4803921568627451,
  (0, 1): 0.5196078431372549},
 (1, 75): {(1, 1): 0.6097560975609756,
  (0, 1): 0.3902439024390244,
  (0, 0): 0.5862068965517241,
  (1, 0): 0.41379310344827586},
 (2, 25): {(1, 0): 0.27049180327868855,
  (1, 1): 0.717948717948718,
  (0, 1): 0.28205128205128205,
  (0, 0): 0.7295081967213115},
 (2, 50): {(1, 1): 0.8778625954198473,
  (0, 1): 0.12213740458015267,
  (0, 0): 0.8611111111111112,
  (1, 0): 0.1388888888888889},
 (2, 75): {(1, 1): 0.8083333333333333,
  (0, 1): 0.19166666666666668,
  (0, 0): 0.8067226890756303,
  (1, 0): 0.19327731092436976},
 (3, 25): {(1, 0): 0.4915254237288136,
  (1, 1): 0.525,
  (0, 1): 0.475,
```

```
 (0, 0): 0.5084745762711864},
(3, 50): {(1, 0): 0.36923076923076925,
 (0, 1): 0.44036697247706424,
 (0, 0): 0.6307692307692307,
 (1, 1): 0.5596330275229358},
(3, 75): {(1, 1): 0.591304347826087,
 (0, 1): 0.40869565217391307,
 (0, 0): 0.6290322580645161,
 (1, 0): 0.3709677419354839},
(4, 25): {(0, 1): 0.7540983606557377,
 (0, 0): 0.21367521367521367,
 (1, 0): 0.7863247863247863,
 (1, 1): 0.2459016393442623},
(4, 50): {(1, 0): 0.8632478632478633,
 (0, 1): 0.8278688524590164,
 (0, 0): 0.13675213675213677,
 (1, 1): 0.1721311475409836},
(4, 75): {(1, 0): 0.7478260869565218,
 (0, 1): 0.6935483870967742,
 (1, 1): 0.3064516129032258,
 (0, 0): 0.25217391304347825},
(5, 25): {(0, 1): 0.5040650406504065,
 (0, 0): 0.47413793103448276,
 (1, 0): 0.5258620689655172,
 (1, 1): 0.4959349593495935},
(5, 50): {(1, 0): 0.5583333333333333,
 (1, 1): 0.4369747899159664,
 (0, 1): 0.5630252100840336,
 (0, 0): 0.44166666666666665},
(5, 75): {(0, 1): 0.5423728813559322,
 (0, 0): 0.47107438016528924,
 (1, 0): 0.5289256198347108,
 (1, 1): 0.4576271186440678},
(6, 25): {(1, 0): 0.9586776859504132,
 (0, 1): 0.9745762711864406,
 (1, 1): 0.025423728813559324,
 (0, 0): 0.04132231404958678},
(6, 50): {(1, 0): 0.9915966386554622,
 (0, 1): 0.9833333333333333,
 (0, 0): 0.008403361344537815,
 (1, 1): 0.016666666666666666},
(6, 75): {(0, 1): 0.9661016949152542,
 (1, 0): 0.95,
 (1, 1): 0.03389830508474576,
 (0, 0): 0.05},
(7, 25): {(1, 0): 0.9159663865546218,
 (0, 1): 0.9310344827586207,
```

```
 (1, 1): 0.06896551724137931,
 (0, 0): 0.08403361344537816},
(7, 50): {(0, 1): 0.957983193277311,
 (1, 0): 0.957983193277311,
 (1, 1): 0.04201680672268908,
 (0, 0): 0.04201680672268908},
(7, 75): {(0, 1): 0.9316239316239316,
 (1, 0): 0.923728813559322,
 (1, 1): 0.06837606837606838,
 (0, 0): 0.07627118644067797},
(8, 25): {(0, 0): 0.9917355371900827,
 (1, 0): 0.008264462809917356,
 (1, 1): 1.0},
(8, 50): {(0, 0): 0.9916666666666667,
 (1, 0): 0.008333333333333333,
 (1, 1): 1.0},
(8, 75): {(0, 0): 0.9916666666666667,
 (1, 0): 0.008333333333333333,
 (1, 1): 1.0},
(10, 25): {(1, 0): 0.43902439024390244,
 (0, 1): 0.45689655172413796,
 (0, 0): 0.5609756097560976,
 (1, 1): 0.5431034482758621},
(10, 50): {(1, 0): 0.47540983606557374,
 (0, 1): 0.49572649572649574,
 (1, 1): 0.5042735042735043,
 (0, 0): 0.5245901639344263},
(10, 75): {(0, 1): 0.5042016806722689,
 (0, 0): 0.5083333333333333,
 (1, 0): 0.49166666666666664,
 (1, 1): 0.4957983193277311},
(11, 25): {(1, 1): 0.9915966386554622,
 (0, 1): 0.008403361344537815,
 (0, 0): 0.9916666666666667,
 (1, 0): 0.008333333333333333},
(11, 50): {(1, 1): 0.9915254237288136,
 (0, 1): 0.00847457627118644,
 (0, 0): 0.9916666666666667,
 (1, 0): 0.008333333333333333},
(11, 75): {(0, 0): 0.5365853658536586,
 (1, 0): 0.4634146341463415,
 (1, 1): 0.5086206896551724,
 (0, 1): 0.49137931034482757},
(12, 25): {(0, 0): 0.22033898305084745,
 (1, 0): 0.7796610169491526,
 (1, 1): 0.2396694214876033,
 (0, 1): 0.7603305785123967},
```

```
(12, 50): {(1, 0): 0.8739495798319328,
 (0, 1): 0.8739495798319328,
 (1, 1): 0.12605042016806722,
 (0, 0): 0.12605042016806722},
(12, 75): {(0, 1): 0.864406779661017,
 (1, 0): 0.8429752066115702,
 (0, 0): 0.15702479338842976,
 (1, 1): 0.13559322033898305},
(13, 25): {(1, 0): 0.8728813559322034,
 (0, 1): 0.8429752066115702,
 (1, 1): 0.15702479338842976,
 (0, 0): 0.1271186440677966},
(13, 50): {(0, 1): 0.8907563025210085,
 (1, 0): 0.8833333333333333,
 (0, 0): 0.11666666666666667,
 (1, 1): 0.1092436974789916},
(13, 75): {(0, 1): 0.7310924369747899,
 (1, 0): 0.725,
 (1, 1): 0.2689075630252101,
 (0, 0): 0.275},
(14, 25): {(1, 0): 0.7368421052631579,
 (1, 1): 0.336,
 (0, 1): 0.664,
 (0, 0): 0.2631578947368421},
(14, 50): {(0, 1): 0.7317073170731707,
 (1, 0): 0.7672413793103449,
 (0, 0): 0.23275862068965517,
 (1, 1): 0.2682926829268293},
(14, 75): {(1, 0): 0.8305084745762712,
 (0, 1): 0.8099173553719008,
 (1, 1): 0.19008264462809918,
 (0, 0): 0.1694915254237288},
(15, 25): {(1, 0): 0.9833333333333333,
 (0, 1): 0.9915966386554622,
 (0, 0): 0.016666666666666666,
 (1, 1): 0.008403361344537815},
(15, 50): {(0, 1): 1.0,
 (1, 0): 0.9916666666666667,
 (0, 0): 0.008333333333333333},
(15, 75): {(0, 0): 0.5548780487804879,
 (1, 0): 0.4451219512195122,
 (0, 1): 0.9733333333333334,
 (1, 1): 0.02666666666666667},
(16, 25): {(0, 1): 0.9401709401709402,
 (1, 0): 0.9166666666666666,
 (0, 0): 0.08333333333333333,
 (1, 1): 0.05982905982905983},
```

```
 (16, 50): {(1, 0): 0.9916666666666667,
  (0, 1): 1.0,
  (0, 0): 0.008333333333333333},
 (16, 75): {(1, 0): 0.9583333333333334,
  (0, 1): 1.0,
  (0, 0): 0.041666666666666664}}
```

[20]:

```python
# Example usage:
# with columns "subject", "condition", and "choice"
for i in df['sub_number'].unique():
    for p in df['proba'].unique():
        print(f"Probability Distribution for subject {i} and condition {p}:")
        for key, probability in probability_distributions_by_group[(i, p)].
 ↪items():
            print(f'P(C_{key[0]} | C_{key[1]}) = {probability:.2f}')
```

```
Probability Distribution for subject 1 and condition 25:
P(C_1 | C_0) = 0.29
P(C_0 | C_1) = 0.30
P(C_1 | C_1) = 0.70
P(C_0 | C_0) = 0.71
Probability Distribution for subject 1 and condition 50:
P(C_0 | C_0) = 0.61
P(C_1 | C_0) = 0.39
P(C_1 | C_1) = 0.48
P(C_0 | C_1) = 0.52
Probability Distribution for subject 1 and condition 75:
P(C_1 | C_1) = 0.61
P(C_0 | C_1) = 0.39
P(C_0 | C_0) = 0.59
P(C_1 | C_0) = 0.41
Probability Distribution for subject 2 and condition 25:
P(C_1 | C_0) = 0.27
P(C_1 | C_1) = 0.72
P(C_0 | C_1) = 0.28
P(C_0 | C_0) = 0.73
Probability Distribution for subject 2 and condition 50:
P(C_1 | C_1) = 0.88
P(C_0 | C_1) = 0.12
P(C_0 | C_0) = 0.86
P(C_1 | C_0) = 0.14
Probability Distribution for subject 2 and condition 75:
P(C_1 | C_1) = 0.81
P(C_0 | C_1) = 0.19
P(C_0 | C_0) = 0.81
P(C_1 | C_0) = 0.19
Probability Distribution for subject 3 and condition 25:
P(C_1 | C_0) = 0.49
```

```
P(C_1 | C_1) = 0.53
P(C_0 | C_1) = 0.47
P(C_0 | C_0) = 0.51
Probability Distribution for subject 3 and condition 50:
P(C_1 | C_0) = 0.37
P(C_0 | C_1) = 0.44
P(C_0 | C_0) = 0.63
P(C_1 | C_1) = 0.56
Probability Distribution for subject 3 and condition 75:
P(C_1 | C_1) = 0.59
P(C_0 | C_1) = 0.41
P(C_0 | C_0) = 0.63
P(C_1 | C_0) = 0.37
Probability Distribution for subject 4 and condition 25:
P(C_0 | C_1) = 0.75
P(C_0 | C_0) = 0.21
P(C_1 | C_0) = 0.79
P(C_1 | C_1) = 0.25
Probability Distribution for subject 4 and condition 50:
P(C_1 | C_0) = 0.86
P(C_0 | C_1) = 0.83
P(C_0 | C_0) = 0.14
P(C_1 | C_1) = 0.17
Probability Distribution for subject 4 and condition 75:
P(C_1 | C_0) = 0.75
P(C_0 | C_1) = 0.69
P(C_1 | C_1) = 0.31
P(C_0 | C_0) = 0.25
Probability Distribution for subject 5 and condition 25:
P(C_0 | C_1) = 0.50
P(C_0 | C_0) = 0.47
P(C_1 | C_0) = 0.53
P(C_1 | C_1) = 0.50
Probability Distribution for subject 5 and condition 50:
P(C_1 | C_0) = 0.56
P(C_1 | C_1) = 0.44
P(C_0 | C_1) = 0.56
P(C_0 | C_0) = 0.44
Probability Distribution for subject 5 and condition 75:
P(C_0 | C_1) = 0.54
P(C_0 | C_0) = 0.47
P(C_1 | C_0) = 0.53
P(C_1 | C_1) = 0.46
Probability Distribution for subject 6 and condition 25:
P(C_1 | C_0) = 0.96
P(C_0 | C_1) = 0.97
P(C_1 | C_1) = 0.03
P(C_0 | C_0) = 0.04
```

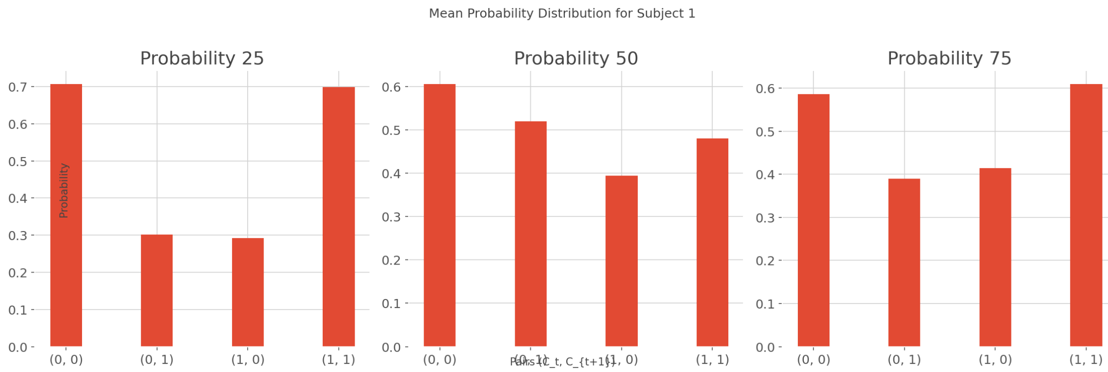
```
Probability Distribution for subject 6 and condition 50:
P(C_1 | C_0) = 0.99
P(C_0 | C_1) = 0.98
P(C_0 | C_0) = 0.01
P(C_1 | C_1) = 0.02
Probability Distribution for subject 6 and condition 75:
P(C_0 | C_1) = 0.97
P(C_1 | C_0) = 0.95
P(C_1 | C_1) = 0.03
P(C_0 | C_0) = 0.05
Probability Distribution for subject 7 and condition 25:
P(C_1 | C_0) = 0.92
P(C_0 | C_1) = 0.93
P(C_1 | C_1) = 0.07
P(C_0 | C_0) = 0.08
Probability Distribution for subject 7 and condition 50:
P(C_0 | C_1) = 0.96
P(C_1 | C_0) = 0.96
P(C_1 | C_1) = 0.04
P(C_0 | C_0) = 0.04
Probability Distribution for subject 7 and condition 75:
P(C_0 | C_1) = 0.93
P(C_1 | C_0) = 0.92
P(C_1 | C_1) = 0.07
P(C_0 | C_0) = 0.08
Probability Distribution for subject 8 and condition 25:
P(C_0 | C_0) = 0.99
P(C_1 | C_0) = 0.01
P(C_1 | C_1) = 1.00
Probability Distribution for subject 8 and condition 50:
P(C_0 | C_0) = 0.99
P(C_1 | C_0) = 0.01
P(C_1 | C_1) = 1.00
Probability Distribution for subject 8 and condition 75:
P(C_0 | C_0) = 0.99
P(C_1 | C_0) = 0.01
P(C_1 | C_1) = 1.00
Probability Distribution for subject 10 and condition 25:
P(C_1 | C_0) = 0.44
P(C_0 | C_1) = 0.46
P(C_0 | C_0) = 0.56
P(C_1 | C_1) = 0.54
Probability Distribution for subject 10 and condition 50:
P(C_1 | C_0) = 0.48
P(C_0 | C_1) = 0.50
P(C_1 | C_1) = 0.50
P(C_0 | C_0) = 0.52
Probability Distribution for subject 10 and condition 75:
```

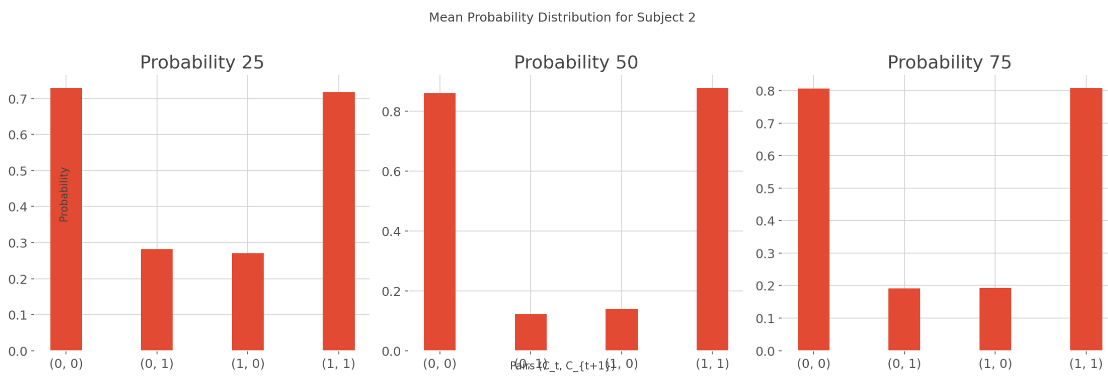
```
P(C_0 | C_1) = 0.50
P(C_0 | C_0) = 0.51
P(C_1 | C_0) = 0.49
P(C_1 | C_1) = 0.50
Probability Distribution for subject 11 and condition 25:
P(C_1 | C_1) = 0.99
P(C_0 | C_1) = 0.01
P(C_0 | C_0) = 0.99
P(C_1 | C_0) = 0.01
Probability Distribution for subject 11 and condition 50:
P(C_1 | C_1) = 0.99
P(C_0 | C_1) = 0.01
P(C_0 | C_0) = 0.99
P(C_1 | C_0) = 0.01
Probability Distribution for subject 11 and condition 75:
P(C_0 | C_0) = 0.54
P(C_1 | C_0) = 0.46
P(C_1 | C_1) = 0.51
P(C_0 | C_1) = 0.49
Probability Distribution for subject 12 and condition 25:
P(C_0 | C_0) = 0.22
P(C_1 | C_0) = 0.78
P(C_1 | C_1) = 0.24
P(C_0 | C_1) = 0.76
Probability Distribution for subject 12 and condition 50:
P(C_1 | C_0) = 0.87
P(C_0 | C_1) = 0.87
P(C_1 | C_1) = 0.13
P(C_0 | C_0) = 0.13
Probability Distribution for subject 12 and condition 75:
P(C_0 | C_1) = 0.86
P(C_1 | C_0) = 0.84
P(C_0 | C_0) = 0.16
P(C_1 | C_1) = 0.14
Probability Distribution for subject 13 and condition 25:
P(C_1 | C_0) = 0.87
P(C_0 | C_1) = 0.84
P(C_1 | C_1) = 0.16
P(C_0 | C_0) = 0.13
Probability Distribution for subject 13 and condition 50:
P(C_0 | C_1) = 0.89
P(C_1 | C_0) = 0.88
P(C_0 | C_0) = 0.12
P(C_1 | C_1) = 0.11
Probability Distribution for subject 13 and condition 75:
P(C_0 | C_1) = 0.73
P(C_1 | C_0) = 0.72
P(C_1 | C_1) = 0.27
```

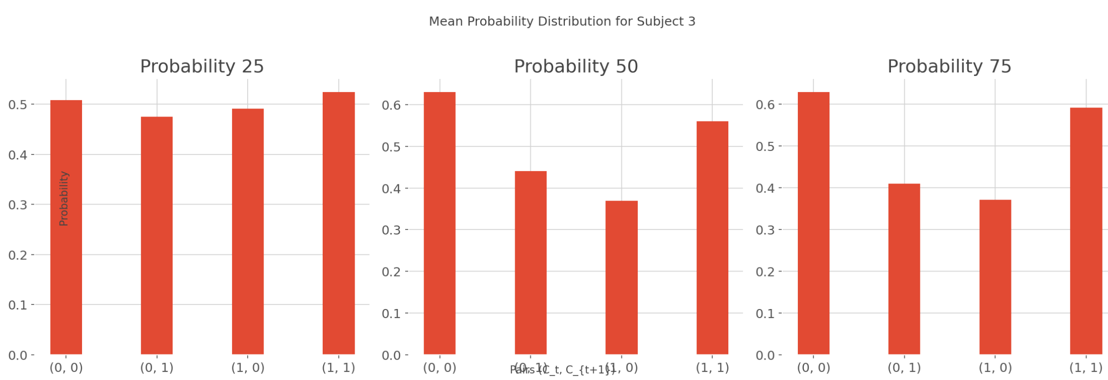
```
P(C_0 | C_0) = 0.28
Probability Distribution for subject 14 and condition 25:
P(C_1 | C_0) = 0.74
P(C_1 | C_1) = 0.34
P(C_0 | C_1) = 0.66
P(C_0 | C_0) = 0.26
Probability Distribution for subject 14 and condition 50:
P(C_0 | C_1) = 0.73
P(C_1 | C_0) = 0.77
P(C_0 | C_0) = 0.23
P(C_1 | C_1) = 0.27
Probability Distribution for subject 14 and condition 75:
P(C_1 | C_0) = 0.83
P(C_0 | C_1) = 0.81
P(C_1 | C_1) = 0.19
P(C_0 | C_0) = 0.17
Probability Distribution for subject 15 and condition 25:
P(C_1 | C_0) = 0.98
P(C_0 | C_1) = 0.99
P(C_0 | C_0) = 0.02
P(C_1 | C_1) = 0.01
Probability Distribution for subject 15 and condition 50:
P(C_0 | C_1) = 1.00
P(C_1 | C_0) = 0.99
P(C_0 | C_0) = 0.01
Probability Distribution for subject 15 and condition 75:
P(C_0 | C_0) = 0.55
P(C_1 | C_0) = 0.45
P(C_0 | C_1) = 0.97
P(C_1 | C_1) = 0.03
Probability Distribution for subject 16 and condition 25:
P(C_0 | C_1) = 0.94
P(C_1 | C_0) = 0.92
P(C_0 | C_0) = 0.08
P(C_1 | C_1) = 0.06
Probability Distribution for subject 16 and condition 50:
P(C_1 | C_0) = 0.99
P(C_0 | C_1) = 1.00
P(C_0 | C_0) = 0.01
Probability Distribution for subject 16 and condition 75:
P(C_1 | C_0) = 0.96
P(C_0 | C_1) = 1.00
P(C_0 | C_0) = 0.04
```

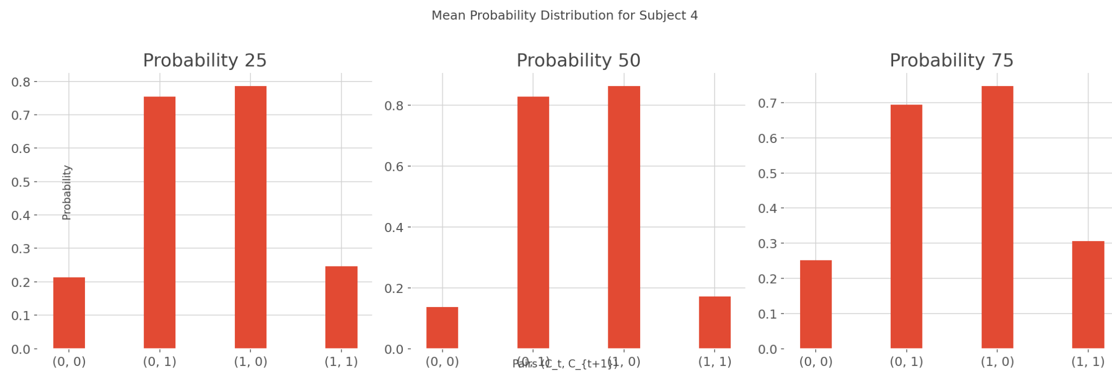
[21]:
```python
# Get unique subjects and probabilities
unique_subjects = df['sub_number'].unique()
unique_probabilities = df['proba'].unique()
```

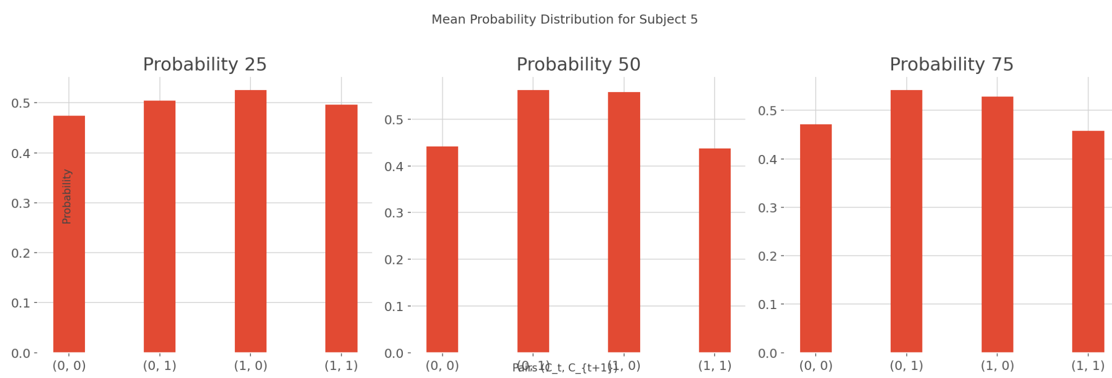
```
# Iterate over subjects
for subject in unique_subjects:
    # Set up subplots for the current subject
    fig, axes = plt.subplots(nrows=1, ncols=len(unique_probabilities),␣
 ↪figsize=(15, 5))

    # Iterate over probabilities
    for j, probability in enumerate(unique_probabilities):
        # Get probability distribution for the current subject and probability
        probability_distribution = probability_distributions_by_group.
 ↪get((subject, probability), {})

        # Get unique pairs and corresponding probabilities
        unique_pairs = sorted(set(pair for pair in probability_distribution.
 ↪keys()))
        probabilities = [probability_distribution.get(pair, 0) for pair in␣
 ↪unique_pairs]

        # Set bar width and offsets
        bar_width = 0.35
        bar_offsets = np.arange(len(unique_pairs))

        # Plot the bar chart
        axes[j].bar(bar_offsets, probabilities, bar_width, label=f'Probability␣
 ↪{probability}')
        axes[j].set_xticks(bar_offsets)
        axes[j].set_xticklabels([f'({pair[0]}, {pair[1]})' for pair in␣
 ↪unique_pairs])
        axes[j].set_title(f'Probability {probability}')

    # Set common labels and legend for the entire figure
    fig.text(0.5, 0.04, 'Pairs (C_t, C_{t+1})', ha='center', va='center')
    fig.text(0.06, 0.5, 'Probability', ha='center', va='center',␣
 ↪rotation='vertical')
    fig.suptitle(f'Mean Probability Distribution for Subject {subject}')

    # Adjust layout
    plt.tight_layout(rect=[0, 0, 1, 0.96])
    plt.show()
```

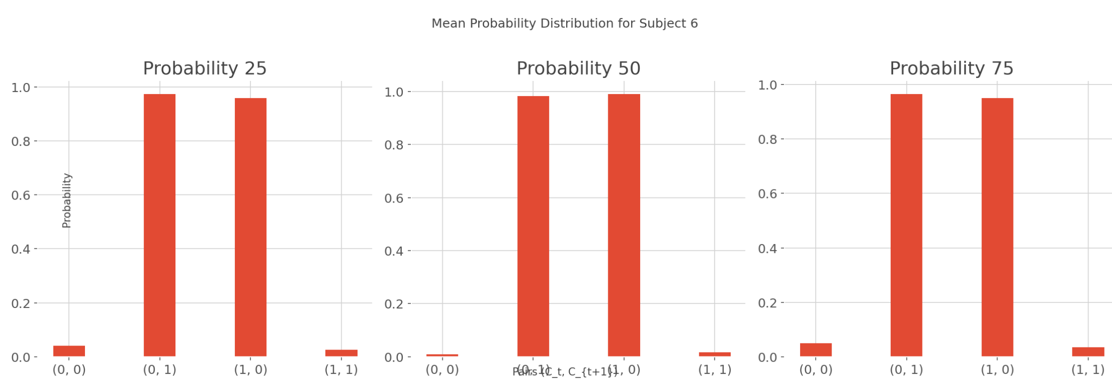

Mean Probability Distribution for Subject 1
Probability 25
Probability 50
Probability 75
Probability
Pairs (C_t, C_{t+1})
(0, 0)
(0, 1)
(1, 0)
(1, 1)

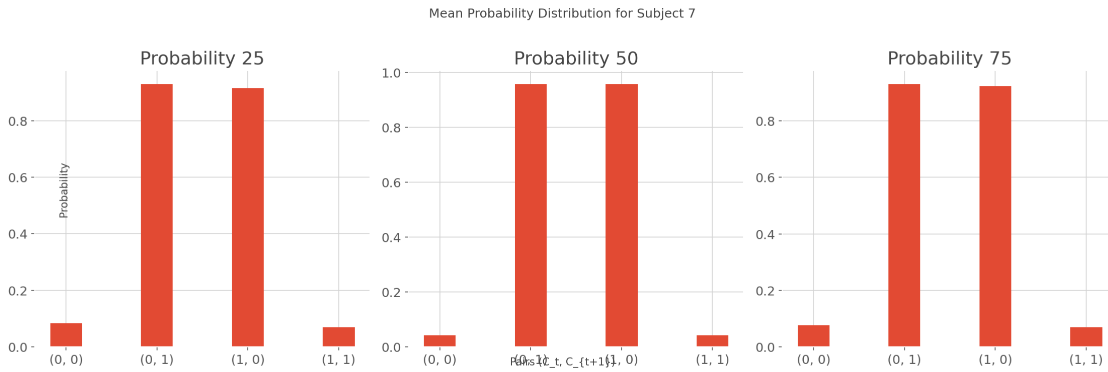

Mean Probability Distribution for Subject 2
Probability 25
Probability 50
Probability 75
Probability
Pairs (C_t, C_{t+1})
(0, 0)
(0, 1)
(1, 0)
(1, 1)

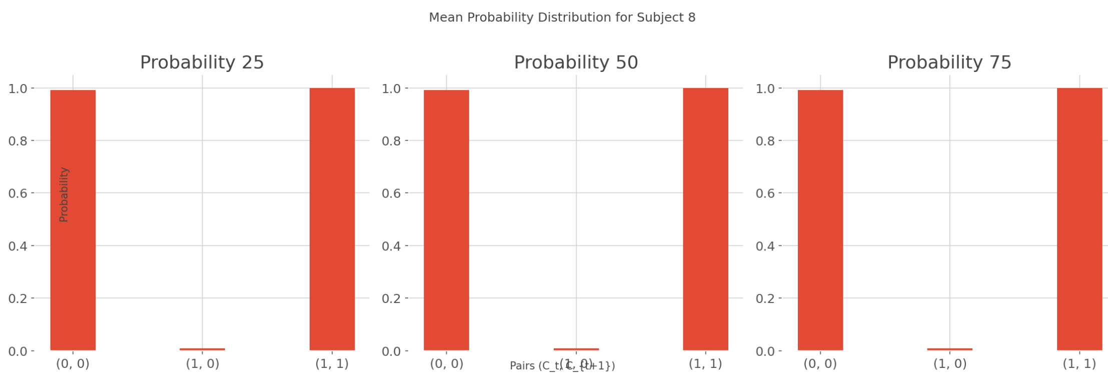

Mean Probability Distribution for Subject 3
Probability 25
Probability 50
Probability 75
Probability
Pairs (C_t, C_{t+1})
(0, 0)
(0, 1)
(1, 0)
(1, 1)

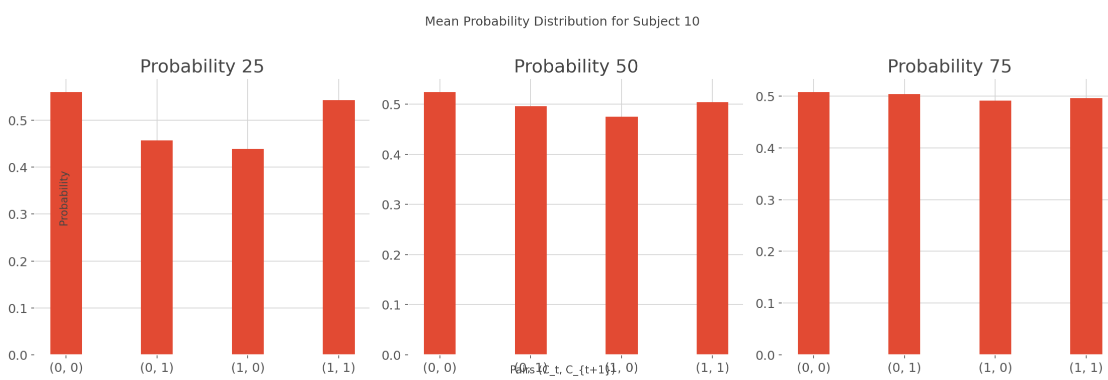

Mean Probability Distribution for Subject 4
Probability 25
Probability 50
Probability 75
Probability
Pairs (C_t, C_{t+1})
(0, 0)
(0, 1)
(1, 0)
(1, 1)

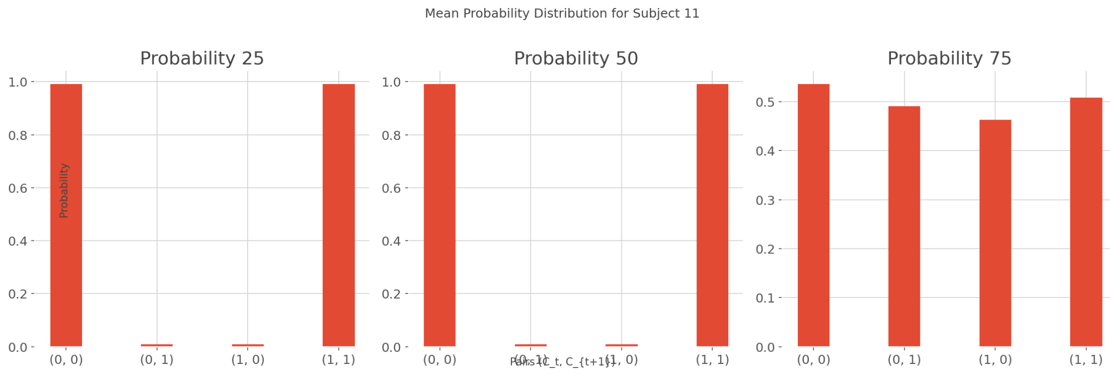

Mean Probability Distribution for Subject 5
Probability 25
Probability 50
Probability 75
Probability
Pairs (C_t, C_{t+1})
(0, 0)
(0, 1)
(1, 0)
(1, 1)

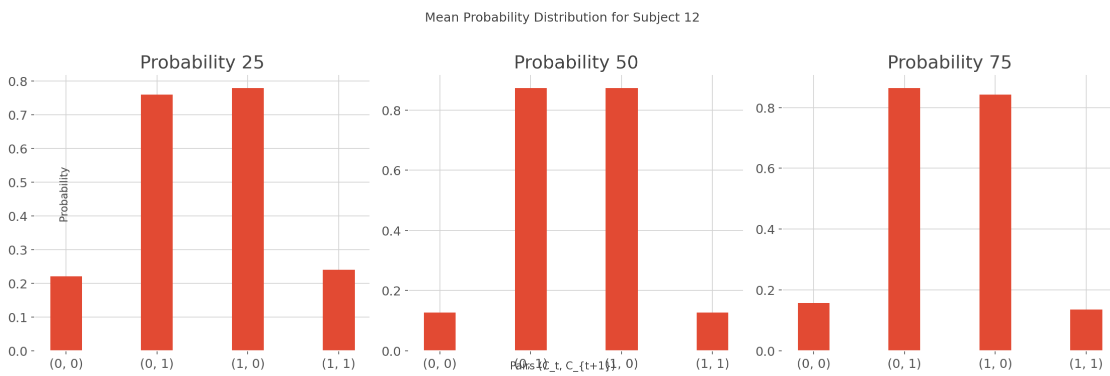

Mean Probability Distribution for Subject 6
Probability 25
Probability 50
Probability 75
Probability
Pairs (C_t, C_{t+1})
(0, 0)
(0, 1)
(1, 0)
(1, 1)

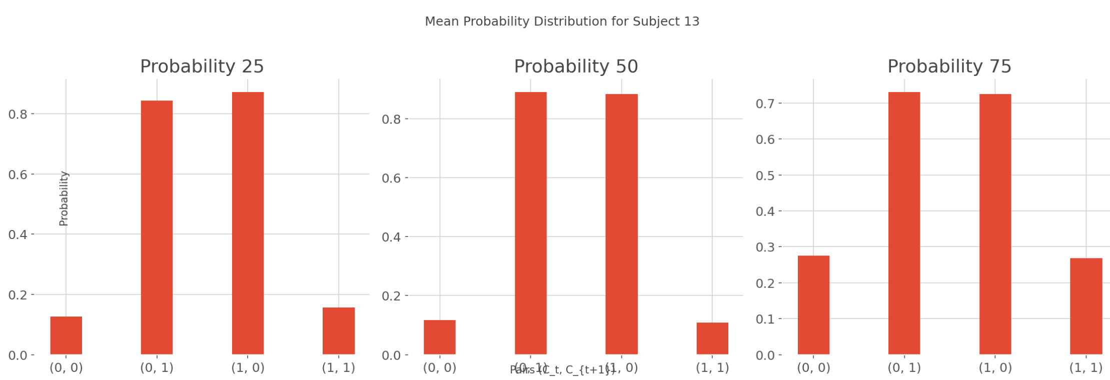

Mean Probability Distribution for Subject 7
Probability 25
Probability 50
Probability 75
Probability
Pairs (C_t, C_{t+1})
(0, 0)
(0, 1)
(1, 0)
(1, 1)

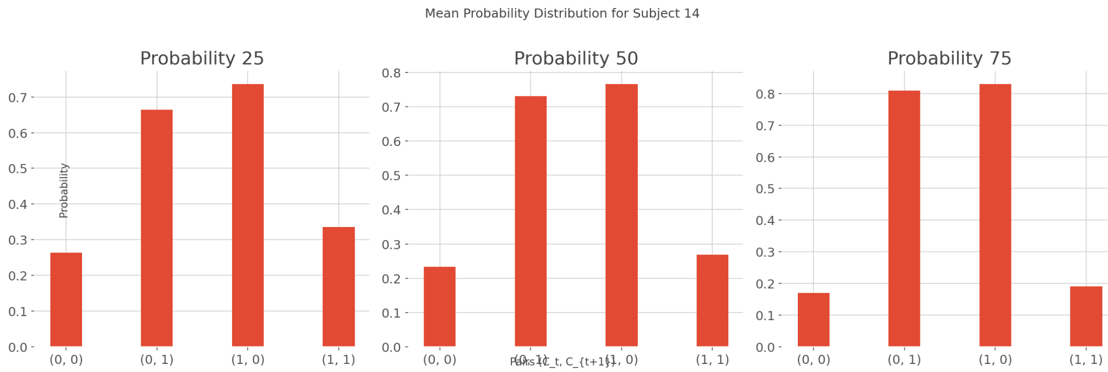

Mean Probability Distribution for Subject 8
Probability 25
Probability 50
Probability 75
Probability
Pairs (C_t, C_{t+1})
(0, 0)
(1, 0)
(1, 1)

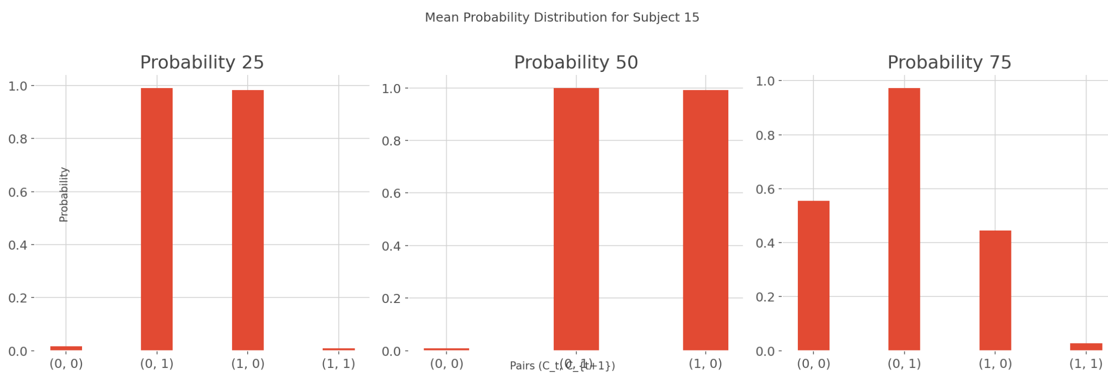

Mean Probability Distribution for Subject 10
Probability 25
Probability 50
Probability 75
Probability
Pairs (C_t, C_{t+1})
(0, 0)
(0, 1)
(1, 0)
(1, 1)

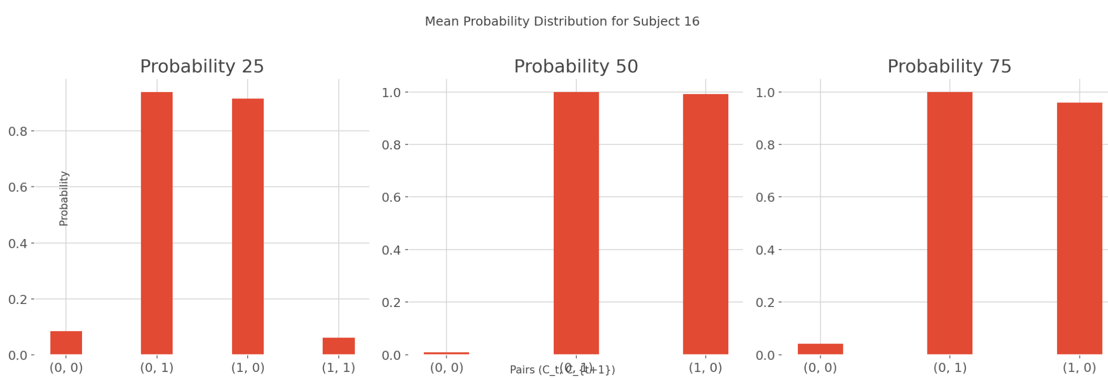

Mean Probability Distribution for Subject 11
Probability 25
Probability 50
Probability 75
Probability
Pairs (C_t, C_{t+1})
(0, 0)
(0, 1)
(1, 0)
(1, 1)


Mean Probability Distribution for Subject 12
Probability 25
Probability 50
Probability 75
Probability
Pairs (C_t, C_{t+1})
(0, 0)
(0, 1)
(1, 0)
(1, 1)


Mean Probability Distribution for Subject 13
Probability 25
Probability 50
Probability 75
Probability
Pairs (C_t, C_{t+1})
(0, 0)
(0, 1)
(1, 0)
(1, 1)


Mean Probability Distribution for Subject 14
Probability 25
Probability 50
Probability 75
Probability
Pairs (C_t, C_{t+1})
(0, 0)
(0, 1)
(1, 0)
(1, 1)


Mean Probability Distribution for Subject 15
Probability 25
Probability 50
Probability 75
Probability
Pairs (C_t, C_{t+1})
(0, 0)
(0, 1)
(1, 0)
(1, 1)

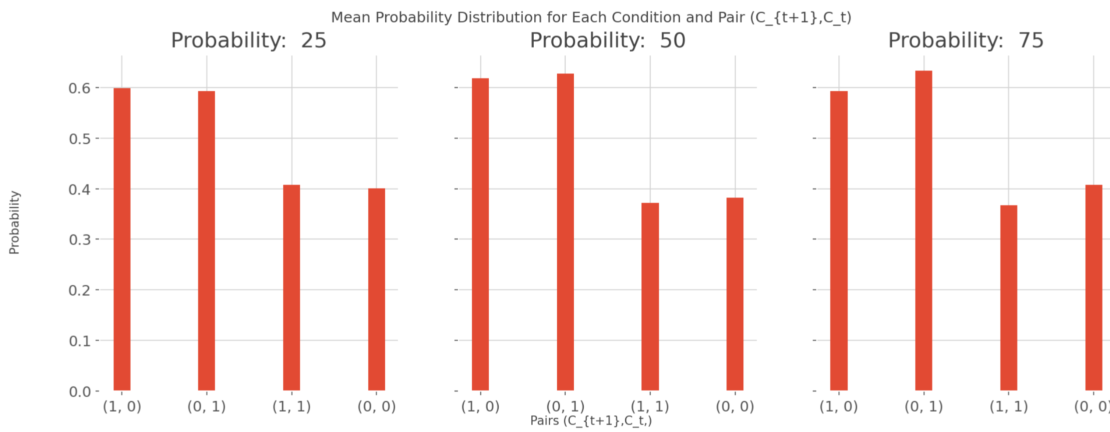

Mean Probability Distribution for Subject 16
Probability 25
Probability 50
Probability 75
Probability
Pairs (C_t, C_{t+1})
(0, 0)
(0, 1)
(1, 0)
(1, 1)

[ ]:
```python
def compute_mean_probability_distribution_tplus1_given_t(dictionary):
    # Create a dictionary to store the mean probability distribution for each␣
 ↪condition
    mean_probability_distribution = {}

    # Iterate over unique conditions
    for (subject, condition), distribution in dictionary.items():
        if condition not in mean_probability_distribution:
            mean_probability_distribution[condition] = Counter()

        mean_probability_distribution[condition].update(distribution)

    # Calculate the mean probability distribution over all subjects for each␣
 ↪condition
    for condition, distribution in mean_probability_distribution.items():
        total_subjects = len(dictionary) // len(mean_probability_distribution)
        mean_probability_distribution[condition] = {key: count / total_subjects␣
 ↪for key, count in distribution.items()}

    return mean_probability_distribution
```

[ ]:
```python
# Assuming you already have the␣
 ↪probability_distributions_by_group_tplus1_given_t dictionary
probability_distributions_by_group_tplus1_given_t =␣
 ↪compute_probability_distribution_tplus1_given_t(df, 'sub_number', 'proba',␣
 ↪'trial_color_chosen')
mean_probability_distribution_tplus1_given_t =␣
 ↪compute_mean_probability_distribution_tplus1_given_t(probability_distributions_by_group_tpl
```

[22]:
```python
# Extract unique pairs (C_t, C_{t+1}) from the first condition (assuming all␣
 ↪conditions have the same pairs)
unique_pairs_t_tplus1 = list(mean_probability_distribution_tplus1_given_t.
 ↪values())[0].keys()

# Prepare data for plotting
num_conditions = len(mean_probability_distribution_tplus1_given_t)
num_pairs = len(unique_pairs_t_tplus1)

# Create subplots
fig, axes = plt.subplots(1, num_conditions, figsize=(15, 5), sharey=True)

bar_width = 0.2
bar_offsets = np.arange(num_pairs)

# Plotting
for idx, (condition, mean_distribution) in␣
 ↪enumerate(mean_probability_distribution_tplus1_given_t.items()):
```

```
    probabilities = [mean_distribution[pair] for pair in unique_pairs_t_tplus1]

    axes[idx].bar(bar_offsets, probabilities, bar_width, label=f'Condition␣
↪{condition}')
    axes[idx].set_xticks(bar_offsets)
    axes[idx].set_xticklabels(unique_pairs_t_tplus1)
    # axes[idx].set_xlabel('Pairs (C_t, C_{t+1})')
    axes[idx].set_title(f'Probability:  {condition}')

# Set common labels and legend
fig.text(0.5, 0.04, 'Pairs (C_{t+1},C_t,)', ha='center', va='center')
fig.text(0.06, 0.5, 'Probability', ha='center', va='center',␣
↪rotation='vertical')
fig.suptitle('Mean Probability Distribution for Each Condition and Pair␣
↪(C_{t+1},C_t)')
# plt.legend()

plt.show()
```



[ ]:
```
"""ooo
Computing P(C_{t+2} | C_{t+1}, C_t)
ooo"""
```

[ ]:
```
from collections import Counter

def compute_probability_distribution_tplus2_given_tplus1_and_t(df, subject_col,␣
↪condition_col, choice_col):
    # df is your DataFrame
    # subject_col is the column name for the subjects
    # condition_col is the column name for the conditions
```

```
    # choice_col is the column name for the choices

    # Create a dictionary to store probability distributions for each subject␣
↪and condition group
    probability_distributions_tplus2_given_tplus1_and_t = {}

    # Iterate over unique subject-condition pairs
    for (subject, condition), group_df in df.groupby([subject_col,␣
↪condition_col]):
        choices = group_df[choice_col].tolist()

        # Count occurrences of each triplet (C_t, C_{t+1}, C_{t+2})
        transition_counts_t_tplus1_tplus2 = Counter(zip(choices[:-2], choices[1:
↪-1], choices[2:]))

        # Compute total counts for each pair (C_{t+1}, C_t)
        total_counts_tplus1_t = Counter(zip(choices[:-1], choices[1:]))

        # Calculate the conditional probabilities for P(C_{t+2} | C_{t+1} & C_t)
        probability_distribution_tplus2_given_tplus1_and_t = {}
        for (choice_t, choice_tplus1, choice_tplus2), count in␣
↪transition_counts_t_tplus1_tplus2.items():
            probability_distribution_tplus2_given_tplus1_and_t[(choice_tplus2,␣
↪choice_tplus1, choice_t)] = count / total_counts_tplus1_t[choice_t,␣
↪choice_tplus1]

        # Store the probability distribution in the dictionary
        probability_distributions_tplus2_given_tplus1_and_t[(subject,␣
↪condition)] = probability_distribution_tplus2_given_tplus1_and_t

    return probability_distributions_tplus2_given_tplus1_and_t
```

[23]:
```
probability_distributions_by_group =␣
↪compute_probability_distribution_tplus2_given_tplus1_and_t(df, 'sub_number',␣
↪'proba', 'trial_color_chosen')
probability_distributions_by_group
```

```
---------------------------------------------------------------------------

--
NameError                                 Traceback (most recent call
last)
Cell In[37], line 1
----> 1 probability_distributions_by_group =
compute_probability_distribution_tplus2_given_tplus1_and_t(df,
'sub_number',
'proba',
'trial_color_chosen')
```

```
      2 probability_distributions_by_group

NameError: name
'compute_probability_distribution_tplus2_given_tplus1_and_t' is not defined
```

[ ]:
```
for i in df['sub_number'].unique():
    for p in df['proba'].unique():
        print(f"Probability Distribution for subject {i} and condition {p}:")
        for key, probability in probability_distributions_by_group[(i, p)].
↪items():
            print(f'P(C_{key[0]} | C_{key[1]},C_{key[2]}) = {probability:.2f}')
```

[ ]:
```
unique_triplets = list(probability_distributions_by_group.values())[0].keys()
unique_triplets
```

[ ]:
```
probability_distributions_by_group.items()
```

[ ]:
```
# Extract unique triplets from the first condition (assuming all conditions␣
↪have the same triplets)

# Prepare data for plotting
num_conditions = len(probability_distributions_by_group)
num_triplets = len(unique_triplets)

# Create subplots
fig, axes = plt.subplots(1, num_conditions, figsize=(15, 5), sharey=True)

bar_width = 0.2
bar_offsets = np.arange(num_triplets)


# Plotting
for idx, (condition, mean_distribution) in␣
↪enumerate(probability_distributions_by_group.items()):
    subject = condition[0]
    condition_value = condition[1]

    print(f"Subject: {subject}, Condition: {condition_value}")
    print(f"Keys in mean_distribution: {mean_distribution.keys()}")

    probabilities = [mean_distribution[triplet] for triplet in unique_triplets]

    axes[idx].bar(bar_offsets, probabilities, bar_width, label=f'Subject␣
↪{subject}, Condition {condition_value}')
    axes[idx].set_xticks(bar_offsets)
    axes[idx].set_xticklabels(unique_triplets, fontsize=8)
    axes[idx].set_title(f'Subject {subject}, Condition {condition_value}')
```

```
# Set common labels
fig.text(0.5, 0.04, 'Triplets (C_{t+2}, C_{t+1}, C_t))', ha='center',␣
↪va='center', fontsize=20)
fig.text(0.06, 0.5, 'Probability', ha='center', va='center',␣
↪rotation='vertical')
fig.suptitle('Mean Probability Distribution for P(C_t+2 | C_t+1, C_t)',␣
↪fontsize=30)

plt.show()
```

[ ]:
```
def compute_mean_probability_distribution(dictionary):
    # Create a dictionary to store the mean probability distribution for each␣
↪condition
    mean_probability_distribution = {}

    # Iterate over unique conditions
    for condition in set(key[1] for key in dictionary.keys()):
        mean_distribution = Counter()
        num_participants = 0

        # Aggregate distributions for the same condition
        for (subject, cond) in dictionary.keys():
            if cond == condition:
                distribution = dictionary[(subject, cond)]
                mean_distribution.update(distribution)
                num_participants += 1

        # Calculate the mean by dividing each count by the number of␣
↪participants
        mean_distribution = {key: count / num_participants for key, count in␣
↪mean_distribution.items()}

        # Store the mean distribution for the condition
        mean_probability_distribution[condition] = mean_distribution

    return mean_probability_distribution
```

[ ]:
```
meanOverSubjects=compute_mean_probability_distribution(probability_distributions_by_group)
meanOverSubjects
```

[ ]:
```
for p in df['proba'].unique():
    print(f"Probability Distribution for proba {p}:")
    for key, probability in meanOverSubjects[(p)].items():
        print(f'P(C_{key[0]} | C_{key[1]},C_{key[2]}) = {probability:.2f}')
```

[ ]:
```
# Extract unique triplets from the first condition (assuming all conditions␣
↪have the same triplets)
unique_triplets = list(meanOverSubjects.values())[0].keys()

# Prepare data for plotting
num_conditions = len(meanOverSubjects)
num_triplets = len(unique_triplets)

# Create subplots
fig, axes = plt.subplots(1, num_conditions, figsize=(15, 5), sharey=True)

bar_width = 0.2
bar_offsets = np.arange(num_triplets)

# Plotting
for idx, (condition, mean_distribution) in enumerate(meanOverSubjects.items()):
    probabilities = [mean_distribution[triplet] for triplet in unique_triplets]

    axes[idx].bar(bar_offsets, probabilities, bar_width, label=f'Condition␣
↪{condition}')
    axes[idx].set_xticks(bar_offsets)
    axes[idx].set_xticklabels(unique_triplets, fontsize=8)
    # axes[idx].set_xlabel('Triplets (C_t, C_{t+1}, C_{t+2})')
    axes[idx].set_title(f'Condition {condition}')

# Set common labels and legend
fig.text(0.5, 0.04, 'Triplets (C_{t+2}, C_{t+1}, C_t))', ha='center',␣
↪va='center',fontsize=20)
fig.text(0.06, 0.5, 'Probability', ha='center', va='center',␣
↪rotation='vertical')
fig.suptitle('Mean Probability Distribution for P(C_t+2 | C_t+1,␣
↪C_t)',fontsize=30)
plt.legend()

plt.show()
```

[ ]:
```
unique_sub_numbers = df['sub_number'].unique()
# Custom color palette for 'color' categories
custom_palette = {'green': 'green', 'red': 'red'}
for sub_number_value in unique_sub_numbers:
    subset_df = df[df['sub_number'] == sub_number_value

    # Set up subplots for each proba
    fig, axes = plt.subplots(nrows=1, ncols=len(subset_df['proba'].unique()),␣
↪figsize=(15, 5), sharey=True)

    # Plot each subplot
```

```
    for i, proba_value in enumerate(subset_df['proba'].unique()):
        proba_subset_df = subset_df[subset_df['proba'] == proba_value]
        ax = axes[i]

        # Group by both "proba" and "color" and compute rolling mean with a␣
↪window of 20
        rolling_mean = proba_subset_df.groupby(["proba", "color"])["meanVelo"].
↪rolling(window=10, min_periods=1).mean().reset_index(level=[0, 1], drop=True)

        # Plot the rolling mean with color as a hue
        sns.lineplot(x="trial_number", y=rolling_mean, hue="color",␣
↪data=proba_subset_df, ax=ax, palette=custom_palette, markers=True)

        ax.set_title(f'sub_number = {sub_number_value}, proba = {proba_value}')
        ax.set_xlabel('Trial Number')
        ax.set_ylabel('Mean Velocity (Rolling Average)')
        ax.legend(title='Color', loc='upper right')

    # Adjust layout for subplots for each subject
    plt.tight_layout()
    plt.show()
```

[ ]:
```
rolling_mean
```

[ ]:
```
#Score of percistency
```

[24]:
```
probability_distributions_by_group_tplus1_given_t.keys ()
```

[24]:
```
dict_keys([(1, 25), (1, 50), (1, 75), (2, 25), (2, 50), (2, 75), (3, 25), (3,
50), (3, 75), (4, 25), (4, 50), (4, 75), (5, 25), (5, 50), (5, 75), (6, 25), (6,
50), (6, 75), (7, 25), (7, 50), (7, 75), (8, 25), (8, 50), (8, 75), (10, 25),
(10, 50), (10, 75), (11, 25), (11, 50), (11, 75), (12, 25), (12, 50), (12, 75),
(13, 25), (13, 50), (13, 75), (14, 25), (14, 50), (14, 75), (15, 25), (15, 50),
(15, 75), (16, 25), (16, 50), (16, 75)])
```

[25]:
```
probability_distributions_by_group_tplus1_given_t
```

[25]:
```
{(1, 25): {(1, 0): 0.2926829268292683,
  (0, 1): 0.3017241379310345,
  (1, 1): 0.6982758620689655,
  (0, 0): 0.7073170731707317},
 (1, 50): {(0, 0): 0.6058394160583942,
  (1, 0): 0.39416058394160586,
  (1, 1): 0.4803921568627451,
```

```
 (0, 1): 0.5196078431372549},
(1, 75): {(1, 1): 0.6097560975609756,
 (0, 1): 0.3902439024390244,
 (0, 0): 0.5862068965517241,
 (1, 0): 0.41379310344827586},
(2, 25): {(1, 0): 0.27049180327868855,
 (1, 1): 0.717948717948718,
 (0, 1): 0.28205128205128205,
 (0, 0): 0.7295081967213115},
(2, 50): {(1, 1): 0.8778625954198473,
 (0, 1): 0.12213740458015267,
 (0, 0): 0.8611111111111112,
 (1, 0): 0.1388888888888889},
(2, 75): {(1, 1): 0.8083333333333333,
 (0, 1): 0.19166666666666668,
 (0, 0): 0.8067226890756303,
 (1, 0): 0.19327731092436976},
(3, 25): {(1, 0): 0.4915254237288136,
 (1, 1): 0.525,
 (0, 1): 0.475,
 (0, 0): 0.5084745762711864},
(3, 50): {(1, 0): 0.36923076923076925,
 (0, 1): 0.44036697247706424,
 (0, 0): 0.6307692307692307,
 (1, 1): 0.5596330275229358},
(3, 75): {(1, 1): 0.591304347826087,
 (0, 1): 0.40869565217391307,
 (0, 0): 0.6290322580645161,
 (1, 0): 0.3709677419354839},
(4, 25): {(0, 1): 0.7540983606557377,
 (0, 0): 0.21367521367521367,
 (1, 0): 0.7863247863247863,
 (1, 1): 0.2459016393442623},
(4, 50): {(1, 0): 0.8632478632478633,
 (0, 1): 0.8278688524590164,
 (0, 0): 0.13675213675213677,
 (1, 1): 0.1721311475409836},
(4, 75): {(1, 0): 0.7478260869565218,
 (0, 1): 0.6935483870967742,
 (1, 1): 0.3064516129032258,
 (0, 0): 0.25217391304347825},
(5, 25): {(0, 1): 0.5040650406504065,
 (0, 0): 0.47413793103448276,
 (1, 0): 0.5258620689655172,
 (1, 1): 0.4959349593495935},
(5, 50): {(1, 0): 0.5583333333333333,
 (1, 1): 0.4369747899159664,
```

```
 (0, 1): 0.5630252100840336,
 (0, 0): 0.44166666666666665},
(5, 75): {(0, 1): 0.5423728813559322,
 (0, 0): 0.47107438016528924,
 (1, 0): 0.5289256198347108,
 (1, 1): 0.4576271186440678},
(6, 25): {(1, 0): 0.9586776859504132,
 (0, 1): 0.9745762711864406,
 (1, 1): 0.025423728813559324,
 (0, 0): 0.04132231404958678},
(6, 50): {(1, 0): 0.9915966386554622,
 (0, 1): 0.9833333333333333,
 (0, 0): 0.008403361344537815,
 (1, 1): 0.016666666666666666},
(6, 75): {(0, 1): 0.9661016949152542,
 (1, 0): 0.95,
 (1, 1): 0.03389830508474576,
 (0, 0): 0.05},
(7, 25): {(1, 0): 0.9159663865546218,
 (0, 1): 0.9310344827586207,
 (1, 1): 0.06896551724137931,
 (0, 0): 0.08403361344537816},
(7, 50): {(0, 1): 0.957983193277311,
 (1, 0): 0.957983193277311,
 (1, 1): 0.04201680672268908,
 (0, 0): 0.04201680672268908},
(7, 75): {(0, 1): 0.9316239316239316,
 (1, 0): 0.923728813559322,
 (1, 1): 0.06837606837606838,
 (0, 0): 0.07627118644067797},
(8, 25): {(0, 0): 0.9917355371900827,
 (1, 0): 0.008264462809917356,
 (1, 1): 1.0},
(8, 50): {(0, 0): 0.9916666666666667,
 (1, 0): 0.008333333333333333,
 (1, 1): 1.0},
(8, 75): {(0, 0): 0.9916666666666667,
 (1, 0): 0.008333333333333333,
 (1, 1): 1.0},
(10, 25): {(1, 0): 0.43902439024390244,
 (0, 1): 0.45689655172413796,
 (0, 0): 0.5609756097560976,
 (1, 1): 0.5431034482758621},
(10, 50): {(1, 0): 0.47540983606557374,
 (0, 1): 0.49572649572649574,
 (1, 1): 0.5042735042735043,
 (0, 0): 0.5245901639344263},
```

(10, 75): {(0, 1): 0.5042016806722689,
 (0, 0): 0.5083333333333333,
 (1, 0): 0.49166666666666664,
 (1, 1): 0.4957983193277311},
(11, 25): {(1, 1): 0.9915966386554622,
 (0, 1): 0.008403361344537815,
 (0, 0): 0.9916666666666667,
 (1, 0): 0.008333333333333333},
(11, 50): {(1, 1): 0.9915254237288136,
 (0, 1): 0.00847457627118644,
 (0, 0): 0.9916666666666667,
 (1, 0): 0.008333333333333333},
(11, 75): {(0, 0): 0.5365853658536586,
 (1, 0): 0.4634146341463415,
 (1, 1): 0.5086206896551724,
 (0, 1): 0.49137931034482757},
(12, 25): {(0, 0): 0.22033898305084745,
 (1, 0): 0.7796610169491526,
 (1, 1): 0.2396694214876033,
 (0, 1): 0.7603305785123967},
(12, 50): {(1, 0): 0.8739495798319328,
 (0, 1): 0.8739495798319328,
 (1, 1): 0.12605042016806722,
 (0, 0): 0.12605042016806722},
(12, 75): {(0, 1): 0.864406779661017,
 (1, 0): 0.8429752066115702,
 (0, 0): 0.15702479338842976,
 (1, 1): 0.13559322033898305},
(13, 25): {(1, 0): 0.8728813559322034,
 (0, 1): 0.8429752066115702,
 (1, 1): 0.15702479338842976,
 (0, 0): 0.1271186440677966},
(13, 50): {(0, 1): 0.8907563025210085,
 (1, 0): 0.8833333333333333,
 (0, 0): 0.11666666666666667,
 (1, 1): 0.1092436974789916},
(13, 75): {(0, 1): 0.7310924369747899,
 (1, 0): 0.725,
 (1, 1): 0.2689075630252101,
 (0, 0): 0.275},
(14, 25): {(1, 0): 0.7368421052631579,
 (1, 1): 0.336,
 (0, 1): 0.664,
 (0, 0): 0.2631578947368421},
(14, 50): {(0, 1): 0.7317073170731707,
 (1, 0): 0.7672413793103449,
 (0, 0): 0.23275862068965517,

```
 (1, 1): 0.2682926829268293},
(14, 75): {(1, 0): 0.8305084745762712,
 (0, 1): 0.8099173553719008,
 (1, 1): 0.19008264462809918,
 (0, 0): 0.1694915254237288},
(15, 25): {(1, 0): 0.9833333333333333,
 (0, 1): 0.9915966386554622,
 (0, 0): 0.016666666666666666,
 (1, 1): 0.008403361344537815},
(15, 50): {(0, 1): 1.0,
 (1, 0): 0.9916666666666667,
 (0, 0): 0.008333333333333333},
(15, 75): {(0, 0): 0.5548780487804879,
 (1, 0): 0.4451219512195122,
 (0, 1): 0.9733333333333334,
 (1, 1): 0.02666666666666667},
(16, 25): {(0, 1): 0.9401709401709402,
 (1, 0): 0.9166666666666666,
 (0, 0): 0.08333333333333333,
 (1, 1): 0.05982905982905983},
(16, 50): {(1, 0): 0.9916666666666667,
 (0, 1): 1.0,
 (0, 0): 0.008333333333333333},
(16, 75): {(1, 0): 0.9583333333333334,
 (0, 1): 1.0,
 (0, 0): 0.041666666666666664}}
```

[ ]:
```
tplus1GivenT=pd.DataFrame(probability_distributions_by_group_tplus1_given_t)
```

[26]:
```
tplus1GivenT
```

[26]:
```
           1                             2    …        15        16          
          25        50        75        25    …        75        25        50   
75
1 0  0.292683  0.394161  0.413793  0.270492  …  0.445122  0.916667  0.991667
0.958333
0 1  0.301724  0.519608  0.390244  0.282051  …  0.973333  0.940171  1.000000
1.000000
1 1  0.698276  0.480392  0.609756  0.717949  …  0.026667  0.059829       NaN
NaN
0 0  0.707317  0.605839  0.586207  0.729508  …  0.554878  0.083333  0.008333
0.041667

[4 rows x 45 columns]
```

[ ]:

[27]:
```python
# Convert dictionary to DataFrame
tplus1GivenT = pd.DataFrame.from_dict({(k1, k2): v2 for k1, d in␣
↪probability_distributions_by_group_tplus1_given_t.items() for k2, v2 in d.
↪items()}, orient='index')

# Reset index and rename columns
tplus1GivenT = tplus1GivenT.reset_index().rename(columns={'level_0': 'Group',␣
↪'level_1': 'Distribution', 0: 'Probability'})

print(tplus1GivenT)
```

```
                    index  Probability
0     ((1, 25), (1, 0))     0.292683
1     ((1, 25), (0, 1))     0.301724
2     ((1, 25), (1, 1))     0.698276
3     ((1, 25), (0, 0))     0.707317
4     ((1, 50), (0, 0))     0.605839
..                  …          …
169  ((16, 50), (0, 1))     1.000000
170  ((16, 50), (0, 0))     0.008333
171  ((16, 75), (1, 0))     0.958333
172  ((16, 75), (0, 1))     1.000000
173  ((16, 75), (0, 0))     0.041667

[174 rows x 2 columns]
```

[28]:
```python
tplus1GivenT.columns
```

[28]:
```
Index(['index', 'Probability'], dtype='object')
```

[29]:
```python
# Dictionary to store the sums for each main key
sums_by_main_key = {}

# Iterate through the main keys
for main_key, sub_dict in probability_distributions_by_group_tplus1_given_t.
↪items():
    # Initialize the sum for the current main key
    current_sum = 0
    # Iterate through the sub-dictionary
    for sub_key, probability in sub_dict.items():
        # Extract the sub-key components
        x, y = sub_key
        # Perform the arithmetic operations and update the sum
        if x == 1 and y == 1:
            current_sum += probability
        elif x == 0 and y == 0:
```

```
            current_sum += probability
        elif x == 0 and y == 1:
            current_sum -= probability
        elif x == 1 and y == 0:
            current_sum -= probability
    # Store the sum for the current main key
    sums_by_main_key[main_key] = current_sum

# Print the sums for each main key
for main_key, sum_value in sums_by_main_key.items():
    print(f"Main Key: {main_key}, Sum: {sum_value}")
```

```
Main Key: (1, 25), Sum: 0.8111858704793944
Main Key: (1, 50), Sum: 0.1724631458422785
Main Key: (1, 75), Sum: 0.3919259882253994
Main Key: (2, 25), Sum: 0.8949138293400589
Main Key: (2, 50), Sum: 1.477947413061917
Main Key: (2, 75), Sum: 1.2301120448179272
Main Key: (3, 25), Sum: 0.06694915254237288
Main Key: (3, 50), Sum: 0.380804516584333
Main Key: (3, 75), Sum: 0.4406732117812061
Main Key: (4, 25), Sum: -1.080846293961048
Main Key: (4, 50), Sum: -1.3822334314137592
Main Key: (4, 75), Sum: -0.8827489481065919
Main Key: (5, 25), Sum: -0.05985421923184747
Main Key: (5, 50), Sum: -0.24271708683473392
Main Key: (5, 75), Sum: -0.14259700238128592
Main Key: (6, 25), Sum: -1.8665079142737078
Main Key: (6, 50), Sum: -1.9498599439775912
Main Key: (6, 75), Sum: -1.8322033898305083
Main Key: (7, 25), Sum: -1.6940017386264852
Main Key: (7, 50), Sum: -1.8319327731092436
Main Key: (7, 75), Sum: -1.7107054903665073
Main Key: (8, 25), Sum: 1.9834710743801653
Main Key: (8, 50), Sum: 1.9833333333333334
Main Key: (8, 75), Sum: 1.9833333333333334
Main Key: (10, 25), Sum: 0.20815811606391932
Main Key: (10, 50), Sum: 0.05772733641586103
Main Key: (10, 75), Sum: 0.008263305322128878
Main Key: (11, 25), Sum: 1.9665266106442578
Main Key: (11, 50), Sum: 1.9663841807909606
Main Key: (11, 75), Sum: 0.09041211101766194
Main Key: (12, 25), Sum: -1.0799831909230986
Main Key: (12, 50), Sum: -1.495798319327731
Main Key: (12, 75), Sum: -1.4147639725451744
Main Key: (13, 25), Sum: -1.4317131250875472
Main Key: (13, 50), Sum: -1.5481792717086833
Main Key: (13, 75), Sum: -0.9121848739495798
```

```
Main Key: (14, 25), Sum: -0.8016842105263158
Main Key: (14, 50), Sum: -0.9978973927670312
Main Key: (14, 75), Sum: -1.280851659896344
Main Key: (15, 25), Sum: -1.9498599439775912
Main Key: (15, 50), Sum: -1.9833333333333334
Main Key: (15, 75), Sum: -0.8369105691056911
Main Key: (16, 25), Sum: -1.7136752136752138
Main Key: (16, 50), Sum: -1.9833333333333334
Main Key: (16, 75), Sum: -1.9166666666666667
```

[30]:
```
sums_by_main_key
```

[30]:
```
{(1, 25): 0.8111858704793944,
 (1, 50): 0.1724631458422785,
 (1, 75): 0.3919259882253994,
 (2, 25): 0.8949138293400589,
 (2, 50): 1.477947413061917,
 (2, 75): 1.2301120448179272,
 (3, 25): 0.06694915254237288,
 (3, 50): 0.380804516584333,
 (3, 75): 0.4406732117812061,
 (4, 25): -1.080846293961048,
 (4, 50): -1.3822334314137592,
 (4, 75): -0.8827489481065919,
 (5, 25): -0.05985421923184747,
 (5, 50): -0.24271708683473392,
 (5, 75): -0.14259700238128592,
 (6, 25): -1.8665079142737078,
 (6, 50): -1.9498599439775912,
 (6, 75): -1.8322033898305083,
 (7, 25): -1.6940017386264852,
 (7, 50): -1.8319327731092436,
 (7, 75): -1.7107054903665073,
 (8, 25): 1.9834710743801653,
 (8, 50): 1.9833333333333334,
 (8, 75): 1.9833333333333334,
 (10, 25): 0.20815811606391932,
 (10, 50): 0.05772733641586103,
 (10, 75): 0.008263305322128878,
 (11, 25): 1.9665266106442578,
 (11, 50): 1.9663841807909606,
 (11, 75): 0.09041211101766194,
 (12, 25): -1.0799831909230986,
 (12, 50): -1.495798319327731,
 (12, 75): -1.4147639725451744,
 (13, 25): -1.4317131250875472,
```

```
(13, 50): -1.5481792717086833,
(13, 75): -0.9121848739495798,
(14, 25): -0.8016842105263158,
(14, 50): -0.9978973927670312,
(14, 75): -1.280851659896344,
(15, 25): -1.9498599439775912,
(15, 50): -1.9833333333333334,
(15, 75): -0.8369105691056911,
(16, 25): -1.7136752136752138,
(16, 50): -1.9833333333333334,
(16, 75): -1.9166666666666667}
```

[31]:
```python
# Initialize lists to store data
subjects = []
probabilities = []
persistence_scores = []

# Iterate through the dictionary items
for key, value in sums_by_main_key.items():
    # Extract subject and probability from the key
    subject, probability = key

    # Append data to lists
    subjects.append(subject)
    probabilities.append(probability)
    persistence_scores.append(value)

# Create DataFrame
percistenceScore = pd.DataFrame({
    'Subject': subjects,
    'Probability': probabilities,
    'Persistence Score': persistence_scores
})

# Display DataFrame
print(percistenceScore)
```

```
   Subject  Probability  Persistence Score
0        1           25           0.811186
1        1           50           0.172463
2        1           75           0.391926
3        2           25           0.894914
4        2           50           1.477947
5        2           75           1.230112
6        3           25           0.066949
7        3           50           0.380805
8        3           75           0.440673
9        4           25          -1.080846
```

```
10         4           50          -1.382233
11         4           75          -0.882749
12         5           25          -0.059854
13         5           50          -0.242717
14         5           75          -0.142597
15         6           25          -1.866508
16         6           50          -1.949860
17         6           75          -1.832203
18         7           25          -1.694002
19         7           50          -1.831933
20         7           75          -1.710705
21         8           25           1.983471
22         8           50           1.983333
23         8           75           1.983333
24        10           25           0.208158
25        10           50           0.057727
26        10           75           0.008263
27        11           25           1.966527
28        11           50           1.966384
29        11           75           0.090412
30        12           25          -1.079983
31        12           50          -1.495798
32        12           75          -1.414764
33        13           25          -1.431713
34        13           50          -1.548179
35        13           75          -0.912185
36        14           25          -0.801684
37        14           50          -0.997897
38        14           75          -1.280852
39        15           25          -1.949860
40        15           50          -1.983333
41        15           75          -0.836911
42        16           25          -1.713675
43        16           50          -1.983333
44        16           75          -1.916667
```

[32]:

```
percistenceScore.groupby('Subject')['Persistence Score'].mean()
```

[32]:

```
Subject
1      0.458525
2      1.200991
3      0.296142
4     -1.115276
5     -0.148389
6     -1.882857
7     -1.745547
```

```
8     1.983379
10    0.091383
11    1.341108
12   -1.330182
13   -1.297359
14   -1.026811
15   -1.590035
16   -1.871225
Name: Persistence Score, dtype: float64
```

[33]:
```
learning=df.groupby(['sub_number','color','proba']).meanVelo.mean().
 ↪reset_index()
learning
```

[33]:
```
    sub_number  color  proba  meanVelo
0            1  green     25 -0.004063
1            1  green     50 -0.088449
2            1  green     75 -0.851774
3            1    red     25 -0.281891
4            1    red     50 -0.052357
..         ...    ...    ...       ...
85          16  green     50  0.072125
86          16  green     75 -0.194897
87          16    red     25  0.150033
88          16    red     50 -0.033862
89          16    red     75 -0.349932

[90 rows x 4 columns]
```

[34]:
```
# Group by 'sub_number' and 'color'
grouped = learning.groupby(['sub_number', 'color'])

# Calculate the mean velocity for probability 75 and 25, respectively
mean_velo_75 = grouped.apply(lambda x: x[x['proba'] == 75]['meanVelo'].mean())
mean_velo_25 = grouped.apply(lambda x: x[x['proba'] == 25]['meanVelo'].mean())

# Calculate the difference
difference = np.abs(mean_velo_75 - mean_velo_25)

# Display the result
print(difference)
```

```
sub_number  color
1           green    0.847711
            red      0.478349
2           green    1.631776
```

```
            red      1.420482
3           green    0.178783
            red      0.010465
4           green    0.525799
            red      0.236479
5           green    0.522206
            red      0.483365
6           green    0.176246
            red      0.051398
7           green    0.456816
            red      0.166019
8           green    1.103647
            red      0.081605
10          green    0.264082
            red      0.118026
11          green    0.337724
            red      0.821605
12          green    0.019265
            red      0.120527
13          green    0.149002
            red      0.073398
14          green    0.545027
            red      0.199084
15          green    0.822538
            red      0.198352
16          green    0.337158
            red      0.499965
dtype: float64
```

[ ]:

```
grouped
```

[35]:

```
difference_green=difference.xs('green', level='color')
difference_red=difference.xs('red', level='color')
percistence=percistenceScore.groupby('Subject')['Persistence Score'].mean().
  ↪reset_index()
percistence
```

[35]:

	Subject	Persistence Score
0	1	0.458525
1	2	1.200991
2	3	0.296142
3	4	-1.115276
4	5	-0.148389
5	6	-1.882857
6	7	-1.745547
7	8	1.983379

8	10	0.091383
9	11	1.341108
10	12	-1.330182
11	13	-1.297359
12	14	-1.026811
13	15	-1.590035
14	16	-1.871225

[ ]:
```
percistence['learningScore']=np.mean([difference_green,difference_red], axis=0)
```

[36]:
```
percistence["learningGreen"]=difference_green.values
percistence["learningRed"]=difference_red.values
percistence
```

[36]:

	Subject	Persistence Score	learningScore	learningGreen	learningRed
0	1	0.458525	0.663030	0.847711	0.478349
1	2	1.200991	1.526129	1.631776	1.420482
2	3	0.296142	0.094624	0.178783	0.010465
3	4	-1.115276	0.381139	0.525799	0.236479
4	5	-0.148389	0.502785	0.522206	0.483365
5	6	-1.882857	0.113822	0.176246	0.051398
6	7	-1.745547	0.311417	0.456816	0.166019
7	8	1.983379	0.592626	1.103647	0.081605
8	10	0.091383	0.191054	0.264082	0.118026
9	11	1.341108	0.579665	0.337724	0.821605
10	12	-1.330182	0.069896	0.019265	0.120527
11	13	-1.297359	0.111200	0.149002	0.073398
12	14	-1.026811	0.372056	0.545027	0.199084
13	15	-1.590035	0.510445	0.822538	0.198352
14	16	-1.871225	0.418562	0.337158	0.499965

[ ]:
```
sns.scatterplot(data=percistence, x="Persistence Score", y="learningGreen",␣
 ↪hue="Subject")
```

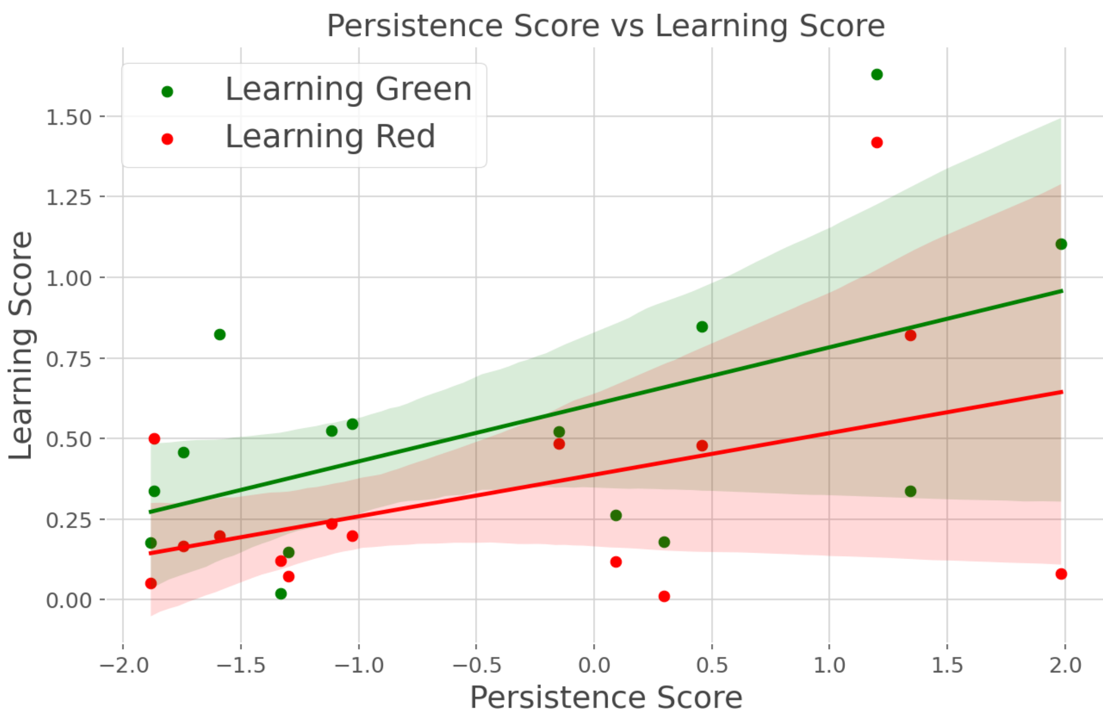
[ ]:
```
sns.scatterplot(data=percistence, x="Persistence Score", y="learningRed",␣
 ↪hue="Subject")
```

[ ]:
```
sns.scatterplot(data=percistence, x="Persistence Score", y="learningScore",␣
 ↪hue="Subject")
```

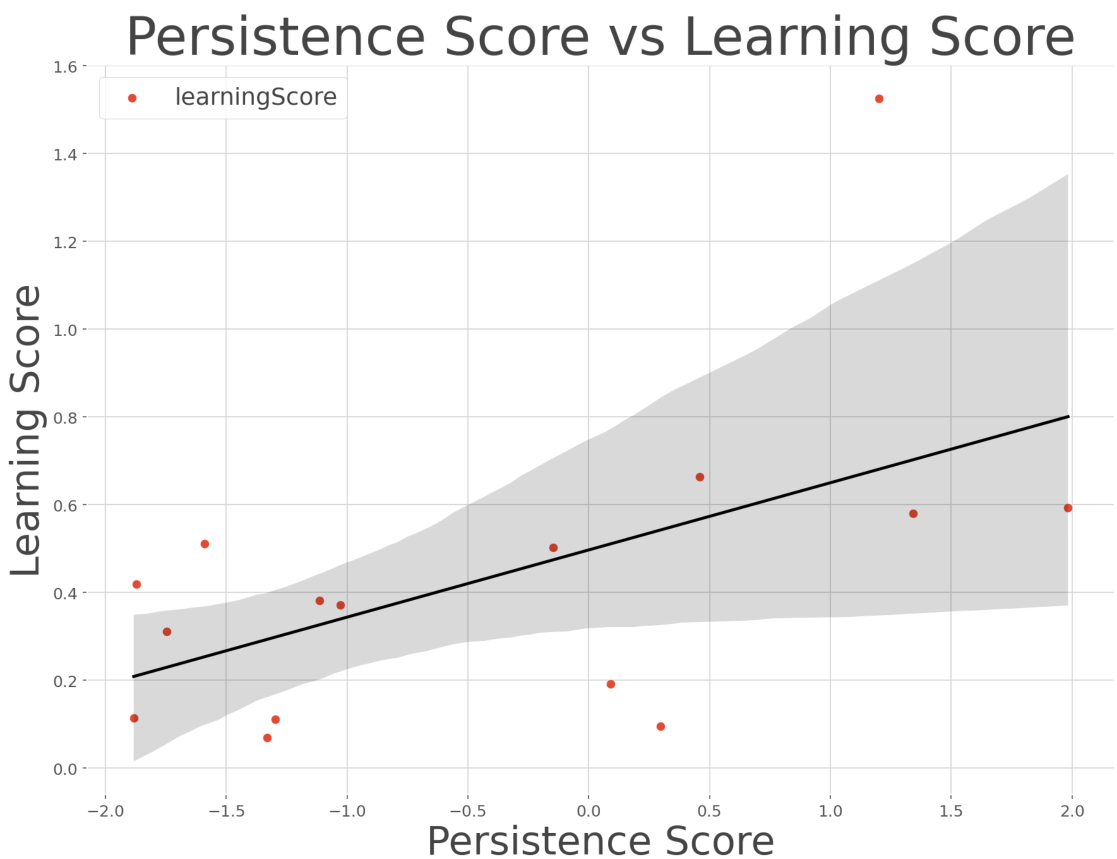
[ ]:
```
# Plotting
plt.figure(figsize=(10, 6))

# Scatter plot for Learning Green
plt.scatter(percistence['Persistence Score'], percistence['learningGreen'],␣
 ↪color='green', label='Learning Green')
```

```
# Scatter plot for Learning Red
plt.scatter(percistence['Persistence Score'], percistence['learningRed'],␣
↪color='red', label='Learning Red')

# Adding labels and title
plt.xlabel('Persistence Score')
plt.ylabel('Learning Score')
plt.title('Persistence Score vs Learning Score')
plt.legend()

# Show plot
plt.show()
```

[37]:
```
# Plotting
plt.figure(figsize=(10, 6))

# Scatter plot for Learning Green
plt.scatter(percistence['Persistence Score'], percistence['learningGreen'],␣
↪color='green', label='Learning Green')

# Scatter plot for Learning Red
plt.scatter(percistence['Persistence Score'], percistence['learningRed'],␣
↪color='red', label='Learning Red')

# Adding fitting lines using seaborn's lmplot
sns.regplot(x='Persistence Score', y='learningGreen', data=percistence,␣
↪scatter=False, color='green')
sns.regplot(x='Persistence Score', y='learningRed', data=percistence,␣
↪scatter=False, color='red')

# Adding labels and title
plt.xlabel('Persistence Score')
plt.ylabel('Learning Score')
plt.title('Persistence Score vs Learning Score')
plt.legend()
plt.savefig('Persistence_Score_vs_Learning_Score.png')
# Show plot
plt.show()
```



[38]:
```python
plt.scatter(data=percistence, x="Persistence Score", y="learningScore")
sns.regplot(x='Persistence Score', y='learningScore', data=percistence,␣
 ↪scatter=False, color='black')
# Adding labels and title
plt.xlabel('Persistence Score',fontsize=30)
plt.ylabel('Learning Score',fontsize=30)
plt.title('Persistence Score vs Learning Score',fontsize=40)
plt.legend()
plt.savefig('Persistence_Score_vs_Learning_Score.png')
# Show plot
plt.show()
```



[39]:
```
import statsmodels.api as sm

# Define the independent variables (Xs) and dependent variables (Ys)
X = percistence[['Persistence Score']]
Y_green = percistence['learningGreen']
Y_red = percistence['learningRed']
Y=percistence['learningScore']
# Add a constant to the independent variables for the intercept term
X = sm.add_constant(X)

# Fit the multiple linear regression models
model_green = sm.OLS(Y_green, X).fit()
model_red = sm.OLS(Y_red, X).fit()
model=sm.OLS(Y, X).fit()
# Print the summary of the regression results
print("Regression Results for Learning Green:")
print(model_green.summary())
```

```python
print("\nRegression Results for Learning Red:")
print(model_red.summary())
```

```
Regression Results for Learning Green:
                            OLS Regression Results
==============================================================================
Dep. Variable:          learningGreen   R-squared:                       0.283
Model:                            OLS   Adj. R-squared:                  0.228
Method:                 Least Squares   F-statistic:                     5.144
Date:                Mon, 11 Mar 2024   Prob (F-statistic):             0.0410
Time:                        12:50:54   Log-Likelihood:                -5.4483
No. Observations:                  15   AIC:                             14.90
Df Residuals:                      13   BIC:                             16.31
Df Model:                           1
Covariance Type:            nonrobust
================================================================================
=====
                        coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
-----
const                 0.6061      0.102      5.914      0.000       0.385
0.828
Persistence Score     0.1770      0.078      2.268      0.041       0.008
0.346
==============================================================================
Omnibus:                        1.545   Durbin-Watson:                   1.828
Prob(Omnibus):                  0.462   Jarque-Bera (JB):                0.698
Skew:                           0.528   Prob(JB):                        0.705
Kurtosis:                       2.973   Cond. No.                         1.56
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.

Regression Results for Learning Red:
                            OLS Regression Results
==============================================================================
Dep. Variable:            learningRed   R-squared:                       0.194
Model:                            OLS   Adj. R-squared:                  0.132
Method:                 Least Squares   F-statistic:                     3.134
Date:                Mon, 11 Mar 2024   Prob (F-statistic):              0.100
Time:                        12:50:54   Log-Likelihood:                -4.4444
No. Observations:                  15   AIC:                             12.89
Df Residuals:                      13   BIC:                             14.30
Df Model:                           1
Covariance Type:            nonrobust
```

```
==============================================================================
=====
                         coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
-----
const                  0.3878      0.096      4.045      0.001       0.181
0.595
Persistence Score      0.1292      0.073      1.770      0.100      -0.028
0.287
==============================================================================
Omnibus:                        6.549   Durbin-Watson:                   2.249
Prob(Omnibus):                  0.038   Jarque-Bera (JB):                3.333
Skew:                           0.894   Prob(JB):                        0.189
Kurtosis:                       4.460   Cond. No.                         1.56
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

[40]:

```python
print("\nRegression Results for Learning Score:")
print(model.summary())
```

```
Regression Results for Learning Score:
                            OLS Regression Results
==============================================================================
Dep. Variable:          learningScore   R-squared:                       0.293
Model:                            OLS   Adj. R-squared:                  0.238
Method:                 Least Squares   F-statistic:                     5.378
Date:                Mon, 11 Mar 2024   Prob (F-statistic):             0.0373
Time:                        12:50:12   Log-Likelihood:                -2.9388
No. Observations:                  15   AIC:                             9.878
Df Residuals:                      13   BIC:                             11.29
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
=====
                         coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
-----
const                  0.4970      0.087      5.732      0.000       0.310
0.684
Persistence Score      0.1531      0.066      2.319      0.037       0.010
0.296
==============================================================================
```

```
Omnibus:                       10.389   Durbin-Watson:                   2.140
Prob(Omnibus):                  0.006   Jarque-Bera (JB):                6.672
Skew:                           1.254   Prob(JB):                       0.0356
Kurtosis:                       5.094   Cond. No.                         1.56
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

[ ]:
```
df_green = df[df.color == 'green']
df_red = df[df.color == 'red']
```

[ ]:
```
df_green
```

[ ]:
```

```

[ ]:
```
df_green
```

[ ]:
```
df_red
```

[ ]:
```
# For df_green
df_green_last_40_trials = df_green.groupby(['sub_number','proba']).tail(40)
df_green_last_40_trials
```

[ ]:
```
# Create a dictionary to map each subject to a specific color
subject_colors = {sub: sns.color_palette('husl',␣
 ↪n_colors=len(df_green['sub_number'].unique()))[i]
                  for i, sub in enumerate(sorted(df_green['sub_number'].
 ↪unique()))}

# Plot mean velocity for each combination of 'sub_number' and 'proba', manually␣
 ↪assigning colors
ax = sns.catplot(x='proba', y='meanVelo', hue='sub_number', kind='point',␣
 ↪data=df_green, palette=subject_colors, legend=False)
plt.xlabel('Probability')
plt.ylabel('Mean Velocity')
plt.title('Mean Velocity across 240 Trials for Each Sub and Proba')

# Get the current axes
ax = plt.gca()

# Manually create legend with subject labels and corresponding colors
handles = [plt.Line2D([0], [0], marker='o', color='w',␣
 ↪markerfacecolor=subject_colors[sub], markersize=10, label=f'Sub {sub}') for␣
 ↪sub in sorted(df_green['sub_number'].unique())]
ax.legend(handles=handles, title='Subject', loc='upper right',fontsize='small')
```

```
plt.show()
```

```
[ ]: # Create a dictionary to map each subject to a specific color
subject_colors = {sub: sns.color_palette('husl',␣
 ↪n_colors=len(df_red['sub_number'].unique()))[i]
                  for i, sub in enumerate(sorted(df_red['sub_number'].
 ↪unique()))}

# Plot mean velocity for each combination of 'sub_number' and 'proba', manually␣
 ↪assigning colors
ax = sns.catplot(x='proba', y='meanVelo', hue='sub_number', kind='point',␣
 ↪data=df_red, palette=subject_colors, legend=False)
plt.xlabel('Probability')
plt.ylabel('Mean Velocity')
plt.title('Mean Velocity across 240 Trials for Each Sub and Proba')

# Get the current axes
ax = plt.gca()

# Manually create legend with subject labels and corresponding colors
handles = [plt.Line2D([0], [0], marker='o', color='w',␣
 ↪markerfacecolor=subject_colors[sub], markersize=10, label=f'Sub {sub}') for␣
 ↪sub in sorted(df_red['sub_number'].unique())]
ax.legend(handles=handles, title='Subject', loc='upper right',fontsize='small')

plt.show()
```

```
[ ]: df_green_last_40_trials.proba.unique()
```

```
[ ]: l_green=df_green_last_40_trials.groupby(["sub_number", "proba"]).meanVelo.
 ↪mean().reset_index()
l_green
```

```
[ ]: df_red_last_40_trials = df_red.groupby(['sub_number','proba']).tail(40)
df_red_last_40_trials
```

```
[ ]: df_red_last_40_trials.proba.unique()
```

```
[ ]: l_red=df_red_last_40_trials.groupby(["sub_number", "proba"]).meanVelo.mean().
 ↪reset_index()
l_red
```

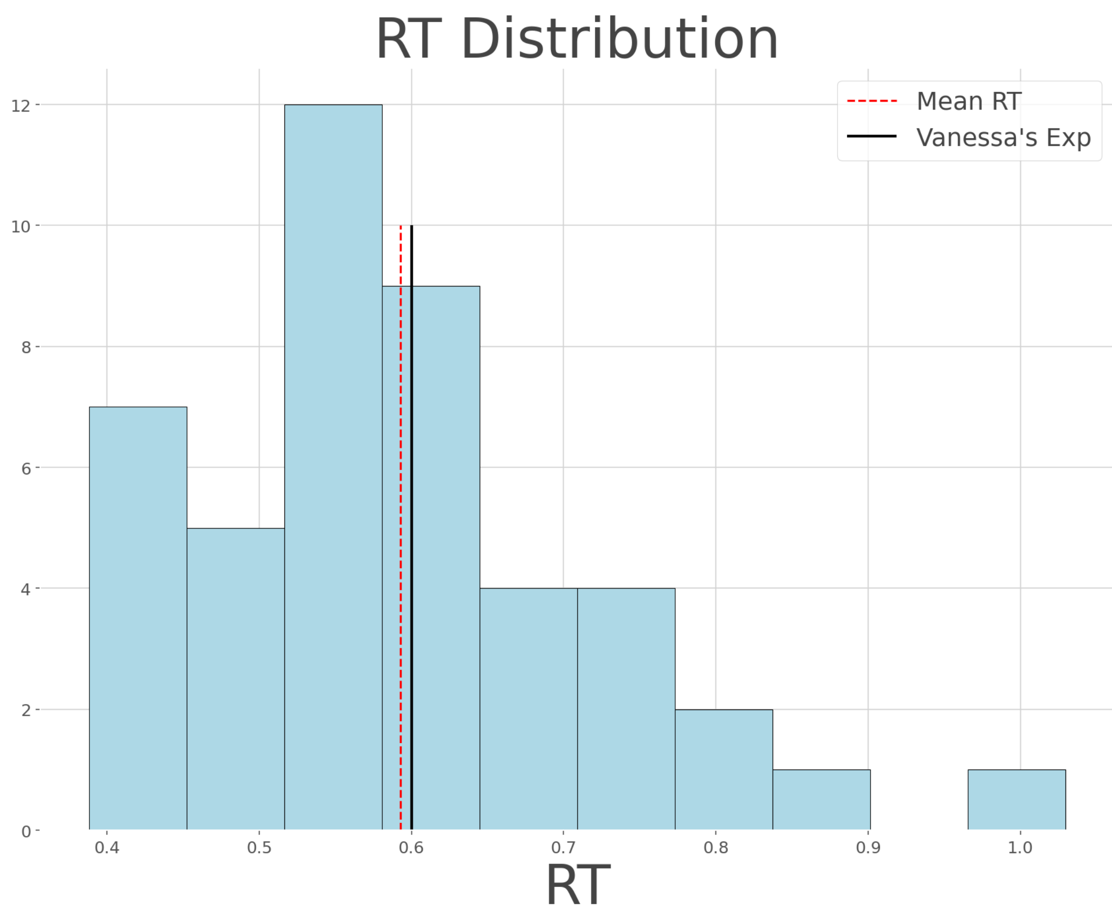
```
[ ]: # Plot the last 40 trials for each color across the 3 probabilities:
# Concatenate the two DataFrames and create a new column 'group' to distinguish␣
 ↪between them
df_green_last_40_trials['color'] = 'Green'
```

```
df_red_last_40_trials['color'] = 'Red'
las40Trials = pd.concat([df_green_last_40_trials, df_red_last_40_trials])

# Plot the boxplot
sns.boxplot(x="proba", y="meanVelo", hue="color", data=las40Trials,␣
 ↪palette={'Green': 'green', 'Red': 'red'})
plt.show()
```

[ ]:
```
df.columns
```

[ ]:
```
df.trial_RT_colochoice
RT=df.groupby(['sub_number','proba']).trial_RT_colochoice.mean().
 ↪reset_index()['trial_RT_colochoice']
```

[41]:
```
plt.hist(RT, color='lightblue', edgecolor='black')
plt.vlines(RT.mean(), 0, 10, color='red', linestyle='--', label='Mean RT')
plt.vlines(0.6, 0, 10, color='black', label='Mean RT',␣
 ↪linewidth=2,label="Vanessa's Exp")
plt.legend()
plt.xlabel('RT',fontsize=40)
plt.title('RT Distribution',fontsize=40)
plt.savefig('RT_Distribution.png')
```



[ ]:
```
df.trial_color_chosen==df.trial_color_UP
```

[ ]:
```
df['arrowChosen']=df.trial_color_chosen==df.trial_color_UP
```

[ ]:
```
df.arrowChosen=['UP' if x==True else 'DOWN' for x in df.arrowChosen]
```

[ ]:
```
df.arrowChosen

df[(df.sub_number==2)&(df.proba==75)]
```

[ ]:
```
df[(df.sub_number==8) & (df.proba==75)]
```

[ ]:
```
df = df.dropna(subset=['meanVelo'])
```

[ ]:
```
df['meanVelo'].isna().sum()
```

[ ]:
```
df.trial_direction
```

[ ]:
```
df[(df.sub_number==16)&(df.proba==75)& (df.arrowChosen=='UP')]
```