stats

May 29, 2025

1 Imports and Constants

[2]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.gridspec import GridSpec
import seaborn as sns
import os


FILENAME_PROCESSED = "processed_data.csv"
DF = pd.read_csv(FILENAME_PROCESSED)
```

2 Plot functions

[3]:

```
def plot_bar_from_column(df, column_name, title, xlabel, ylabel):
    # Count the occurrences of each role

    if column_name == 'bva_familiarity':
        order = ["Not at all familiar", "Somewhat familiar", "Quite familiar",␣
 ↪"Expert"]
        role_counts = df[column_name].value_counts().reindex(order,␣
 ↪fill_value=0)
    else:
        role_counts = df[column_name].value_counts()

    # Set style
    plt.style.use('seaborn-v0_8-whitegrid')
    fig, ax = plt.subplots(figsize=(9, 6))

    # Bar colors and transparency
    bars = ax.bar(role_counts.index, role_counts.values, color='#4c72b0',␣
 ↪alpha=0.85)

    # Add count labels above each bar
    for bar in bars:
        height = bar.get_height()
        ax.text(bar.get_x() + bar.get_width() / 2, height + 0.5,␣
 ↪str(int(height)),
                ha='center', va='bottom', fontsize=11, fontweight='bold')

    # Titles and labels
    ax.set_title(title, fontsize=14, fontweight='bold', pad=20)
    ax.set_ylabel(ylabel, fontsize=12)
    ax.set_xlabel(xlabel, fontsize=12)

    # Rotate x-axis labels for readability
    plt.xticks(rotation=30, ha='right', fontsize=11)
    plt.yticks(fontsize=11)

    # Remove top and right borders
    ax.spines['top'].set_visible(False)
    ax.spines['right'].set_visible(False)

    plt.tight_layout()
    plt.show()

def plot_histogram(df, column_name, title, xlabel, ylabel):
    fig, ax = plt.subplots(figsize=(9, 6))
    # Compute histogram data and get bin edges
    counts, bins, patches = ax.hist(
        df[column_name],
        bins=range(min(df[column_name]), max(df[column_name]) + 2),
        color='#4c72b0',
        alpha=0.85,
    )
    # Titles and labels
    ax.set_title(title, fontsize=14, fontweight='bold')
    ax.set_ylabel(ylabel, fontsize=12)
    ax.set_xlabel(xlabel, fontsize=12)

    # Calculate bin centers for xticks
    bin_centers = [(bins[i] + bins[i+1]) / 2 for i in range(len(bins) - 1)]
    ax.set_xticks(bin_centers)
    ax.set_xticklabels([str(int(center)) for center in bin_centers],␣
 ↪fontsize=11)

    # Remove top and right borders
    ax.spines['top'].set_visible(False)
    ax.spines['right'].set_visible(False)

    plt.tight_layout()
    plt.show()
```

```python
def plot_domain_counts(df):
    # Mapping of column names to display names
    domain_labels = {
        'domain_web': 'Web / Cloud services',
        'domain_mobile': 'Mobile Applications',
        'domain_embedded': 'Embedded / IoT',
        'domain_enterprise': 'Enterprise / Backend',
        'domain_ai': 'AI / ML',
        'domain_games': 'Games'
    }

    # Count the number of 1.0s in each domain column
    counts = {label: int(df[col].sum()) for col, label in domain_labels.items() if col in df.columns}

    # Handle 'domain_other' based on text_domain_other content
    if 'domain_other' in df.columns and 'text_domain_other' in df.columns:
        # Get non-empty entries where domain_other == 1.0
        other_entries = df.loc[df['domain_other'] == 1.0, 'text_domain_other'].dropna().astype(str).str.strip()
        # Count how many filled responses
        other_count = other_entries.shape[0]
        # Join unique values for display (optional)
        other_label = 'Other: ' + ', '.join(other_entries.unique()) if not other_entries.empty else 'Other'
        counts[other_label] = other_count

    # Make sure the color list matches the number of bars
    colors = [
    '#A1C9F4',  # light blue
    '#8DE5A1',  # mint green
    '#FF9F9B',  # coral pink
    '#D0BBFF',  # light lavender
    '#FFBE7D',  # peach
    '#4c72b0',  # dark blue
    '#C7CEEA',  # pale periwinkle
    ]
    color_list = colors[:len(counts)] if len(counts) <= len(colors) else [colors[i % len(colors)] for i in range(len(counts))]


    # Plot
    plt.figure(figsize=(10, 6))
    bars = plt.bar(counts.keys(), counts.values(), color=color_list, alpha=0.9)

    # Add count labels on top of bars
    for bar in bars:
        height = bar.get_height()
        plt.text(bar.get_x() + bar.get_width()/2, height + 0.5, str(int(height)),
                 ha='center', va='bottom', fontsize=11, fontweight='bold')

    plt.title('Distribution of Domains', fontsize=14, fontweight='bold')
    plt.xlabel('Domain', fontsize=12)
    plt.ylabel('Count', fontsize=12)
    plt.xticks(rotation=30, ha='right', fontsize=11)
    plt.yticks(fontsize=11)

    plt.tight_layout()
    plt.show()

def plot_descriptive_comparison(desc_df, sut, category):
    # Filter only mean and std for plotting
    stats_to_plot = desc_df[['mean', 'std']]

    # Plot
    stats_to_plot.plot(kind='bar', figsize=(10, 6), colormap='Pastel1')

    plt.title(f'Mean and Standard Deviation per {sut} example (and all combined)', fontsize=14, fontweight='bold')
    plt.xlabel('Email_n_clear Columns', fontsize=12)
    plt.ylabel('Value', fontsize=12)
    plt.xticks(rotation=45, ha='right')
    plt.grid(axis='y', linestyle='--', alpha=0.7)
    plt.legend(title='Statistic', fontsize=10)
    plt.tight_layout()
    plt.show()


def plot_box_per_sut(df, sut):
    """
    sut: email, bytecnt, date, bmi
    Plots 4 subplots (boxplots) for the 4 categories and saves stats as CSVs.
    """

    categories = ['clarity', 'completeness', 'correctness', 'usefulness']
    fig, axes = plt.subplots(2, 2, figsize=(14, 10))
    fig.suptitle(f'Scores per Example for {sut.upper()} SUT', fontsize=16, fontweight='bold', y=1.02)
    plt.subplots_adjust(hspace=0.4, wspace=0.3)

    for i, category in enumerate(categories):
        desc_stats, cols = calculate_descriptives(df, sut, category)
```

```python
        # Save stats to CSV
        csv_filename = f"{sut}_{category}_stats.csv"
        desc_stats.to_csv(csv_filename)

        # Prepare data for plotting
        melted_df = df[cols].melt(var_name=sut.capitalize(), value_name=category.capitalize())
        combined = pd.DataFrame({
            sut.capitalize(): ['combined'] * df[cols].stack().shape[0],
            category.capitalize(): df[cols].stack().values
        })
        plot_df = pd.concat([melted_df, combined], ignore_index=True)

        # Plot
        ax = axes[i // 2, i % 2]
        sns.boxplot(
            data=plot_df,
            x=f"{sut.capitalize()}",
            y=f"{category.capitalize()}",
            hue=f"{sut.capitalize()}",
            palette='pastel',
            ax=ax,
            legend=False
        )
        ax.set_title(f'{category.capitalize()}', fontsize=12, fontweight='bold')
        ax.set_xlabel('Example')
        ax.set_ylabel('Score')
        ax.set_yticks([1, 2, 3, 4, 5])
        ax.tick_params(axis='x', rotation=30)

    plt.tight_layout()
    plt.show()


def plot_box_and_table_per_category(df, category):
    """
    category: clarity, completeness, correctness, usefulness
    This function plots and calculates stats per SUT for the selected category.
    """

    # Identify SUTs based on column prefixes (assumes consistent naming like email_1_clarity, etc.)
    suts = ['email', 'bytecnt', 'date', 'bmi']
    data = []
    stats_dict = {}
    for sut in suts:
        # Get all columns for this SUT and category
        cols = [col for col in df.columns if col.startswith(f"{sut}_") and col.
↪endswith(f"_{category}")]
        if not cols:
            continue  # Skip if this SUT has no relevant columns

        # Combine all into a long format
        stacked = df[cols].stack().reset_index(drop=True)
        data.extend([(sut.upper(), val) for val in stacked.dropna()])

        # Descriptive stats for this SUT
        sut_desc = stacked.describe()[['mean', 'std', 'min', '25%', '50%',␣
↪'75%', 'max']]
        stats_dict[sut.upper()] = np.round(sut_desc.values, 2)

    # Create DataFrame for plotting
    plot_df = pd.DataFrame(data, columns=['SUT', category.capitalize()])

    # Create DataFrame for stats table
    stats_df = pd.DataFrame.from_dict(stats_dict, orient='index',
                                  columns=['mean', 'std', 'min', '25%',␣
↪'50%', '75%', 'max'])

    # Plot layout: one column, two rows
    fig = plt.figure(constrained_layout=True, figsize=(10, 10))
    gs = GridSpec(2, 1, height_ratios=[3, 1], figure=fig)

    # Boxplot
    ax1 = fig.add_subplot(gs[0])
    sns.boxplot(data=plot_df, x='SUT', y=category.capitalize(),␣
↪palette='pastel', ax=ax1, hue = 'SUT', legend=False)
    ax1.set_title(f'{category.capitalize()} Score Distribution per SUT',␣
↪fontsize=14, fontweight='bold')
    ax1.set_xlabel('SUT')
    ax1.set_ylabel(f'{category.capitalize()}')
    ax1.set_yticks([1, 2, 3, 4, 5])
    ax1.tick_params(axis='x', rotation=30)

    # Table
    ax2 = fig.add_subplot(gs[1])
    ax2.axis('off')
    table = ax2.table(cellText=stats_df.values,
                      rowLabels=stats_df.index,
                      colLabels=stats_df.columns,
                      loc='center')
    table.auto_set_font_size(False)
    table.set_fontsize(10)
    table.scale(1.2, 1.5)
    ax2.set_title('Descriptive Statistics by SUT', fontsize=12,␣
 ↪fontweight='bold')

    plt.show()
```

3 Calculation functions

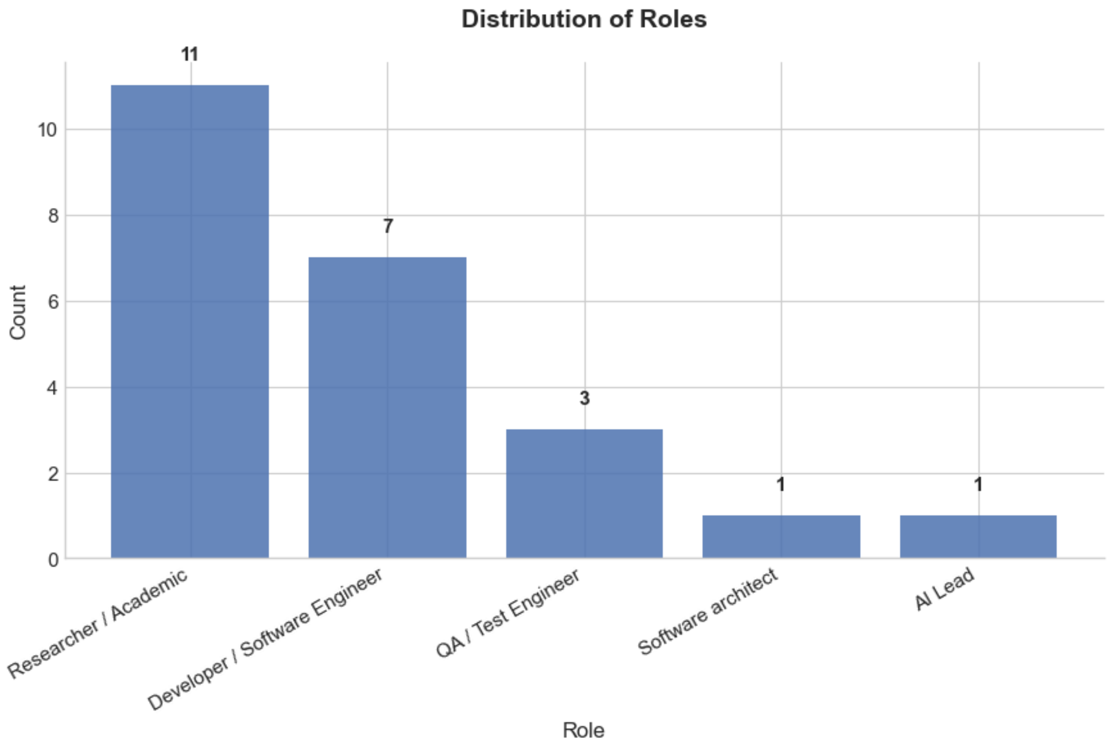
[4]:

```
# Function to calculate descriptive stats
def calculate_descriptives(df, sut, category):
    cols = [col for col in df.columns if col.startswith(sut) and col.
 ↪endswith(category)]
    desc_per_column = df[cols].describe().T[['mean', 'std', 'min', '25%',␣
 ↪'50%', '75%', 'max']]
    combined = pd.concat([df[col].dropna() for col in cols], ignore_index=True)
    combined_desc = combined.describe()[['mean', 'std', 'min', '25%', '50%',␣
 ↪'75%', 'max']]
    desc_per_column.loc['combined'] = combined_desc
    return desc_per_column, cols
```

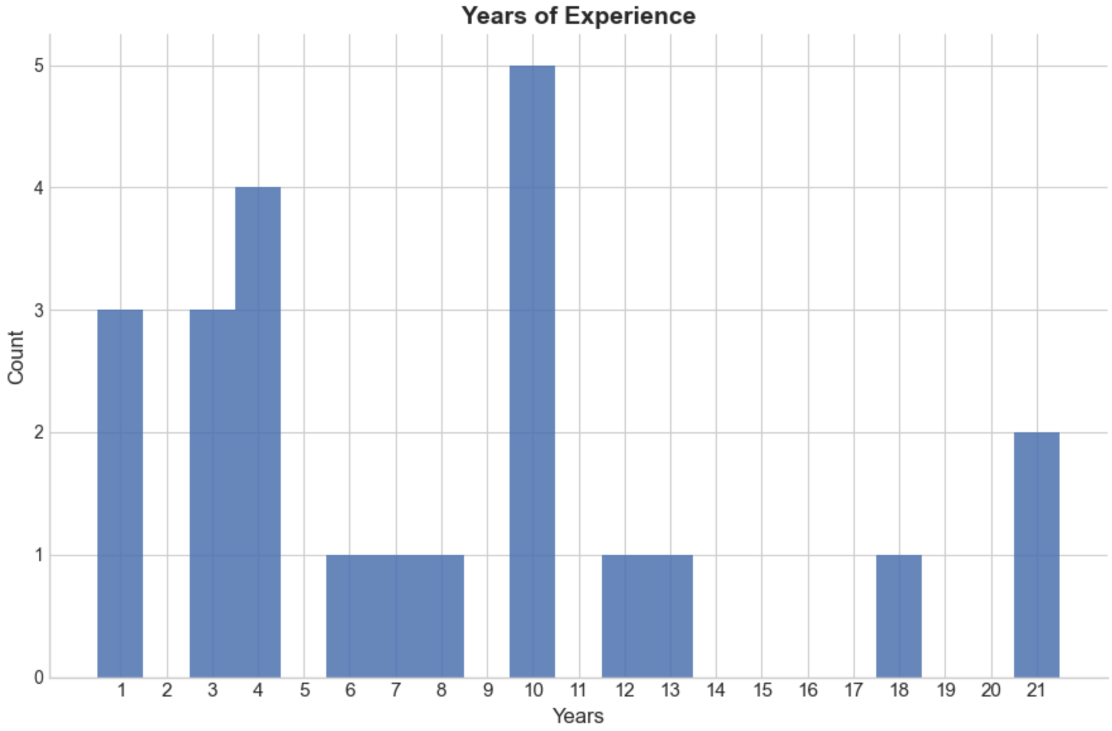
4 Background of Participants

[5]:

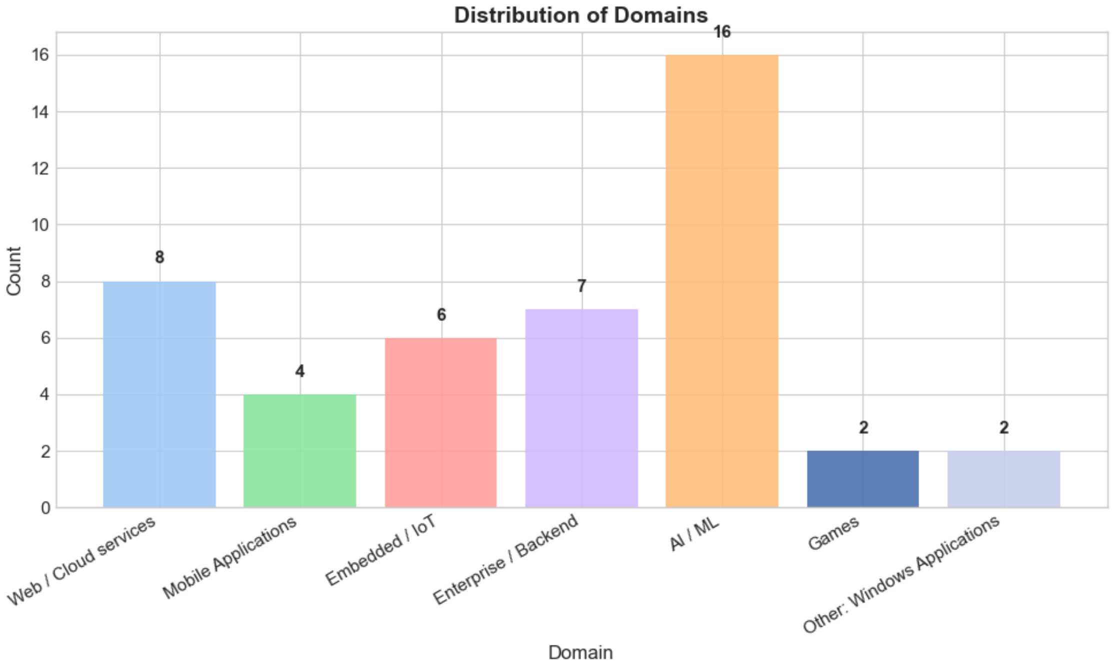
```
plot_bar_from_column(DF, 'role', 'Distribution of Roles', 'Role', 'Count')
```



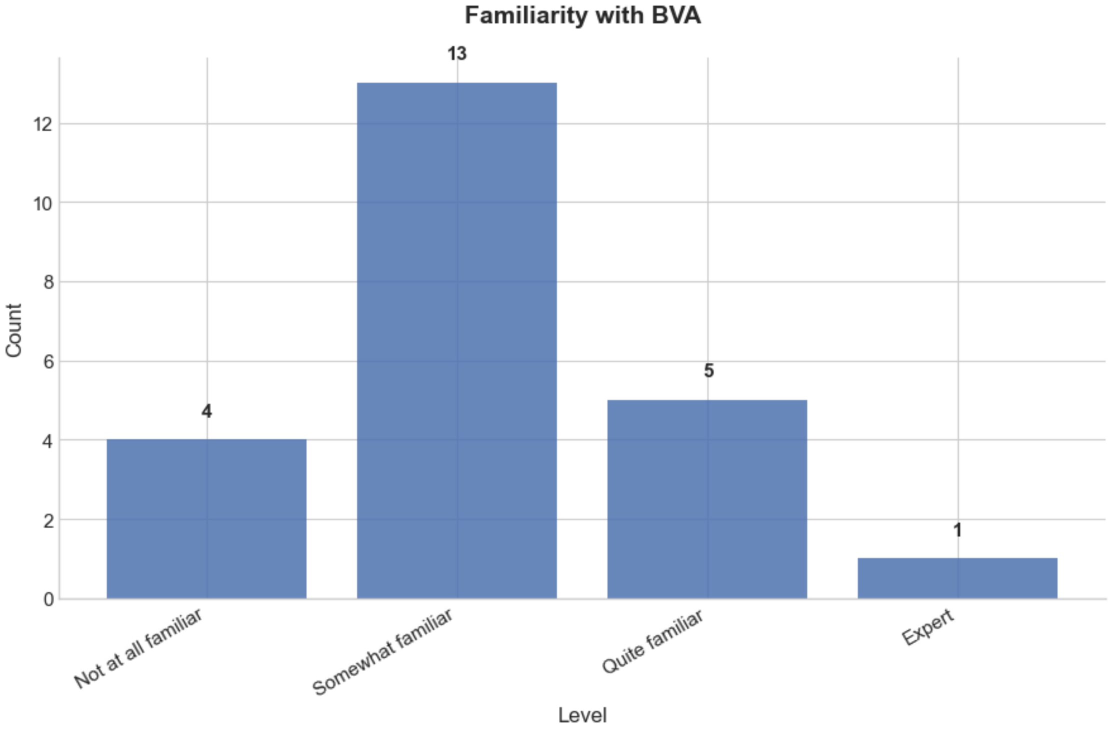
```
[6]: plot_histogram(DF, 'years_experience', 'Years of Experience', 'Years', 'Count')
```



```
[7]: plot_domain_counts(DF)
```



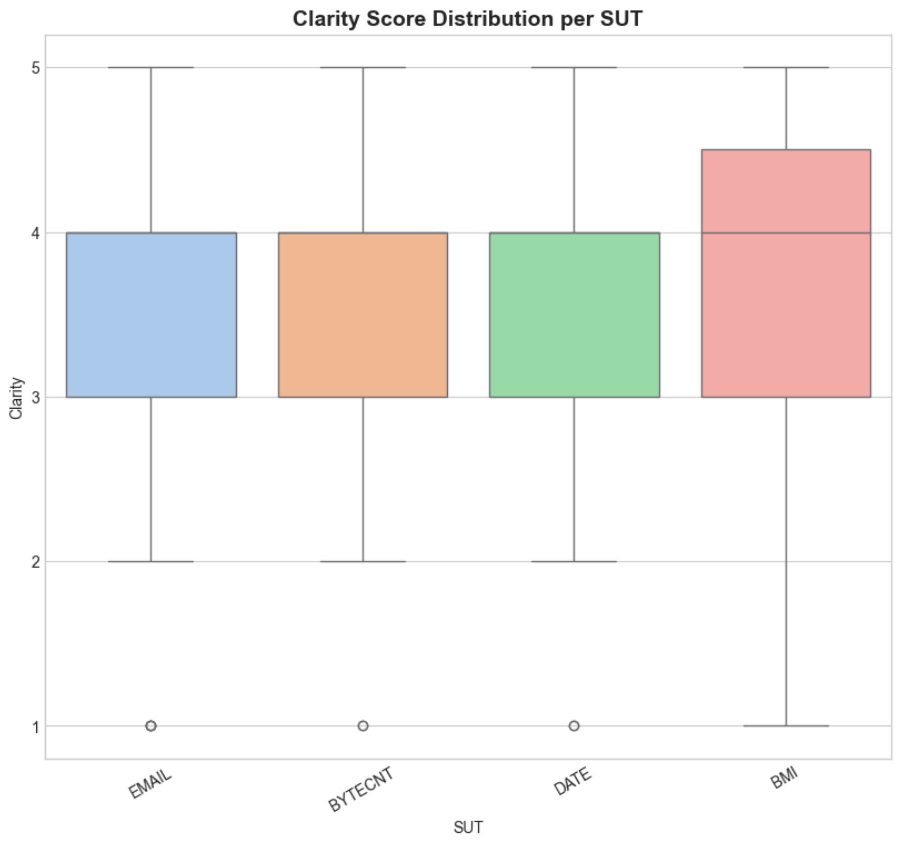
[8]:
```
plot_bar_from_column(DF, 'bva_familiarity', 'Familiarity with BVA', 'Level',␣
↪'Count')
```



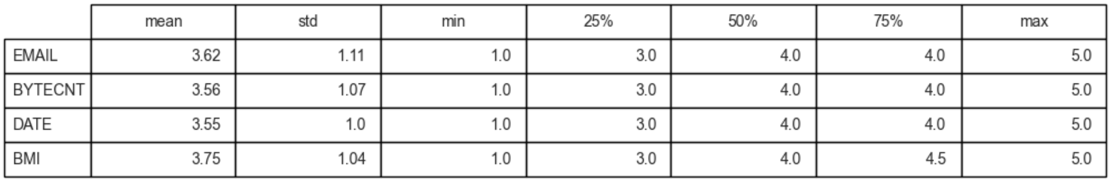
5 Descriptive Statistics: Across all SUTs

5.1 *How does –Clarity– vary across all SUTs?*

[9]:
```
plot_box_and_table_per_category(DF, "clarity")
```



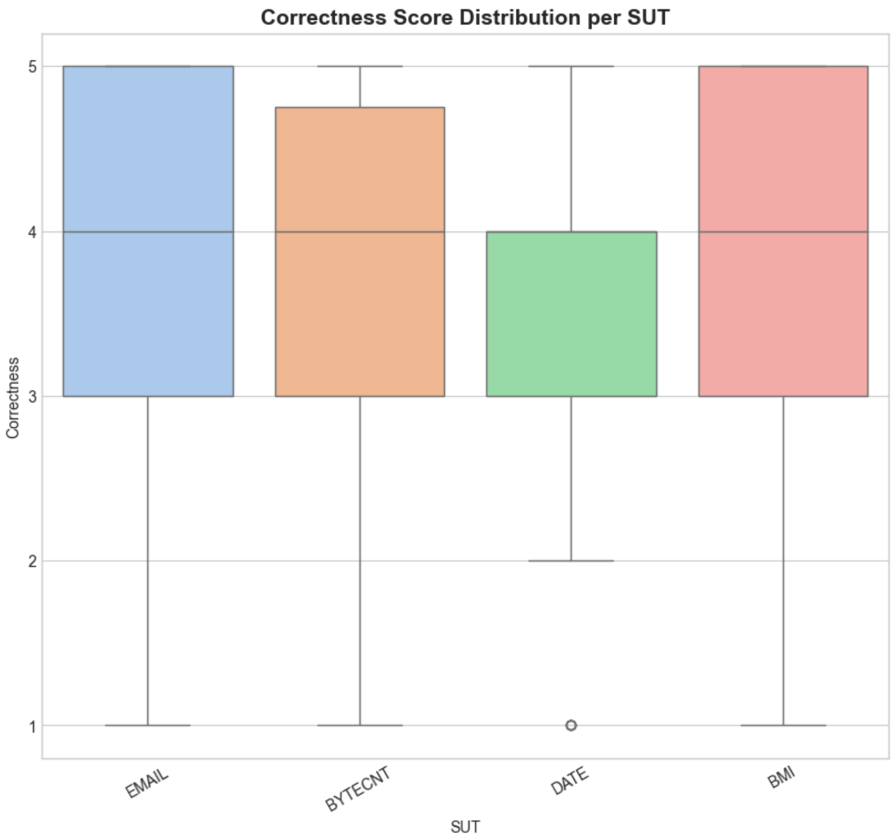
Descriptive Statistics by SUT

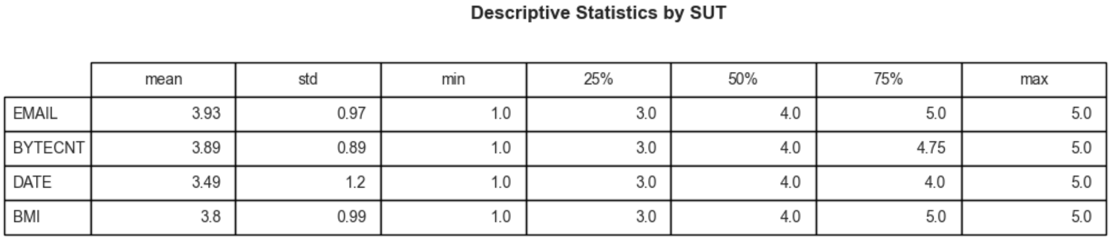


	mean	std	min	25%	50%	75%	max
EMAIL	3.62	1.11	1.0	3.0	4.0	4.0	5.0
BYTECNT	3.56	1.07	1.0	3.0	4.0	4.0	5.0
DATE	3.55	1.0	1.0	3.0	4.0	4.0	5.0
BMI	3.75	1.04	1.0	3.0	4.0	4.5	5.0

5.2 *How does –Correctness– vary across all SUTs?*

[10]:
```
plot_box_and_table_per_category(DF, "correctness")
```



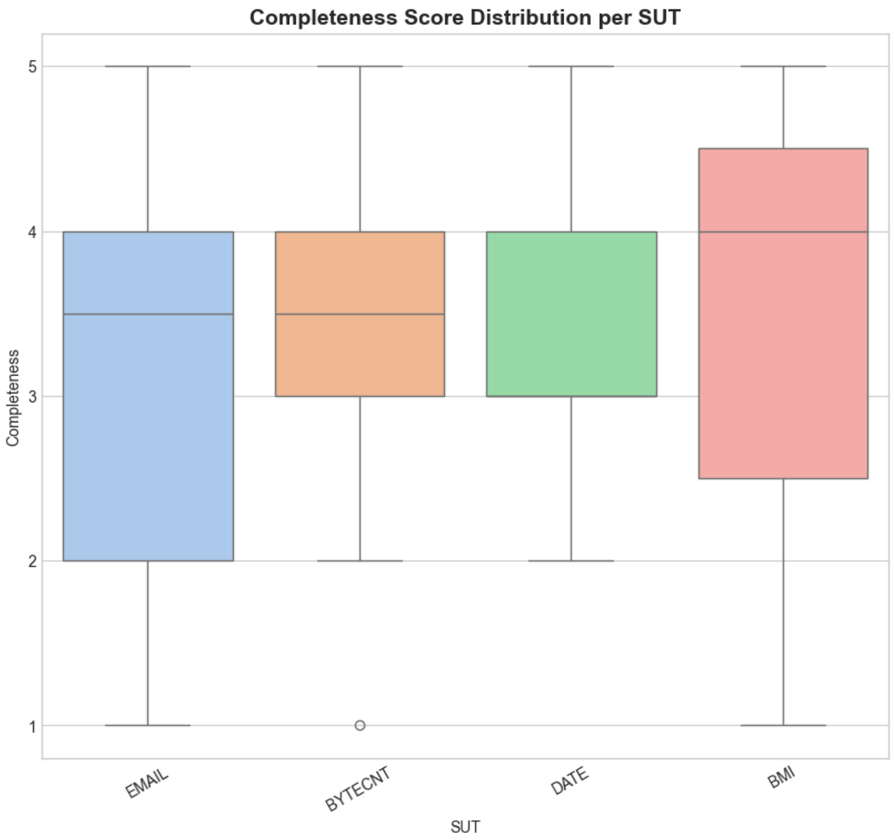


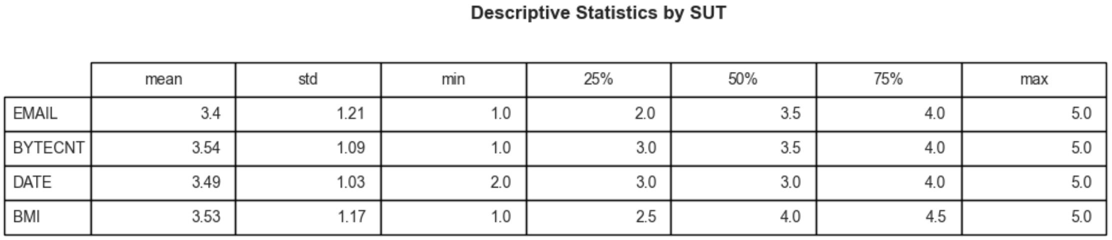
Descriptive Statistics by SUT

	mean	std	min	25%	50%	75%	max
EMAIL	3.93	0.97	1.0	3.0	4.0	5.0	5.0
BYTECNT	3.89	0.89	1.0	3.0	4.0	4.75	5.0
DATE	3.49	1.2	1.0	3.0	4.0	4.0	5.0
BMI	3.8	0.99	1.0	3.0	4.0	5.0	5.0

5.3 *How does –Completeness– vary across all SUTs?*

[11]:
```
plot_box_and_table_per_category(DF, "completeness")
```



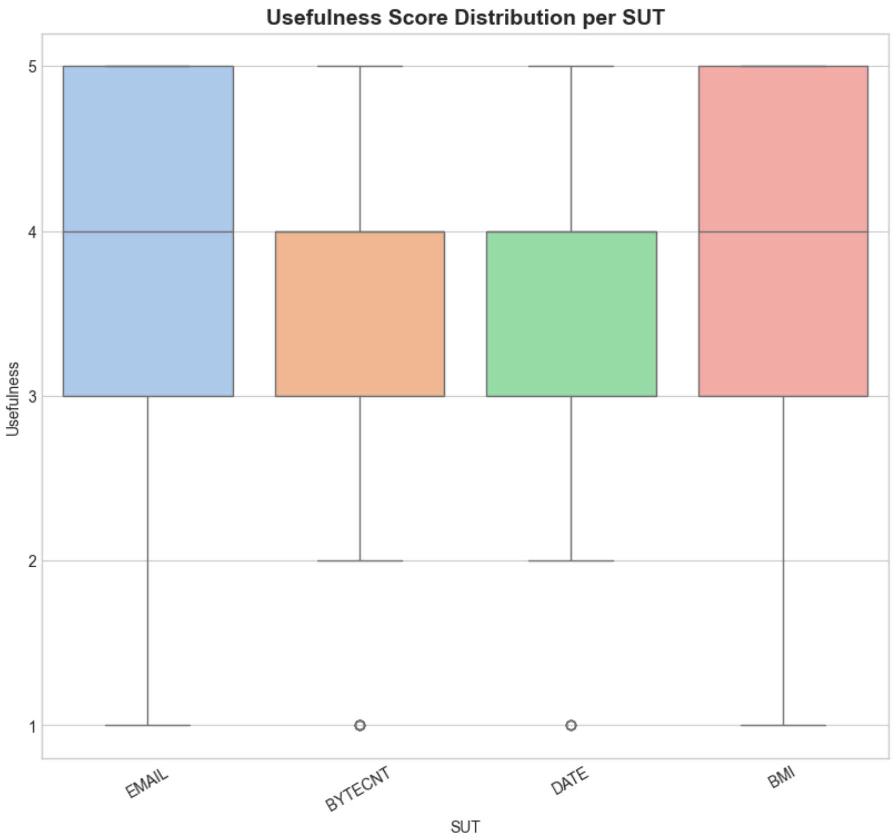


Descriptive Statistics by SUT

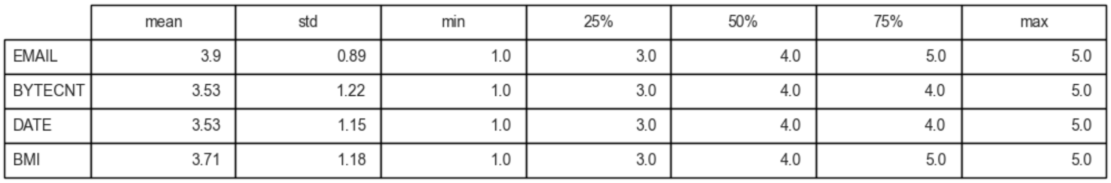
	mean	std	min	25%	50%	75%	max
EMAIL	3.4	1.21	1.0	2.0	3.5	4.0	5.0
BYTECNT	3.54	1.09	1.0	3.0	3.5	4.0	5.0
DATE	3.49	1.03	2.0	3.0	3.0	4.0	5.0
BMI	3.53	1.17	1.0	2.5	4.0	4.5	5.0

5.4 *How does –Usefulness– vary across all SUTs?*

```
[12]: plot_box_and_table_per_category(DF, "usefulness")
```



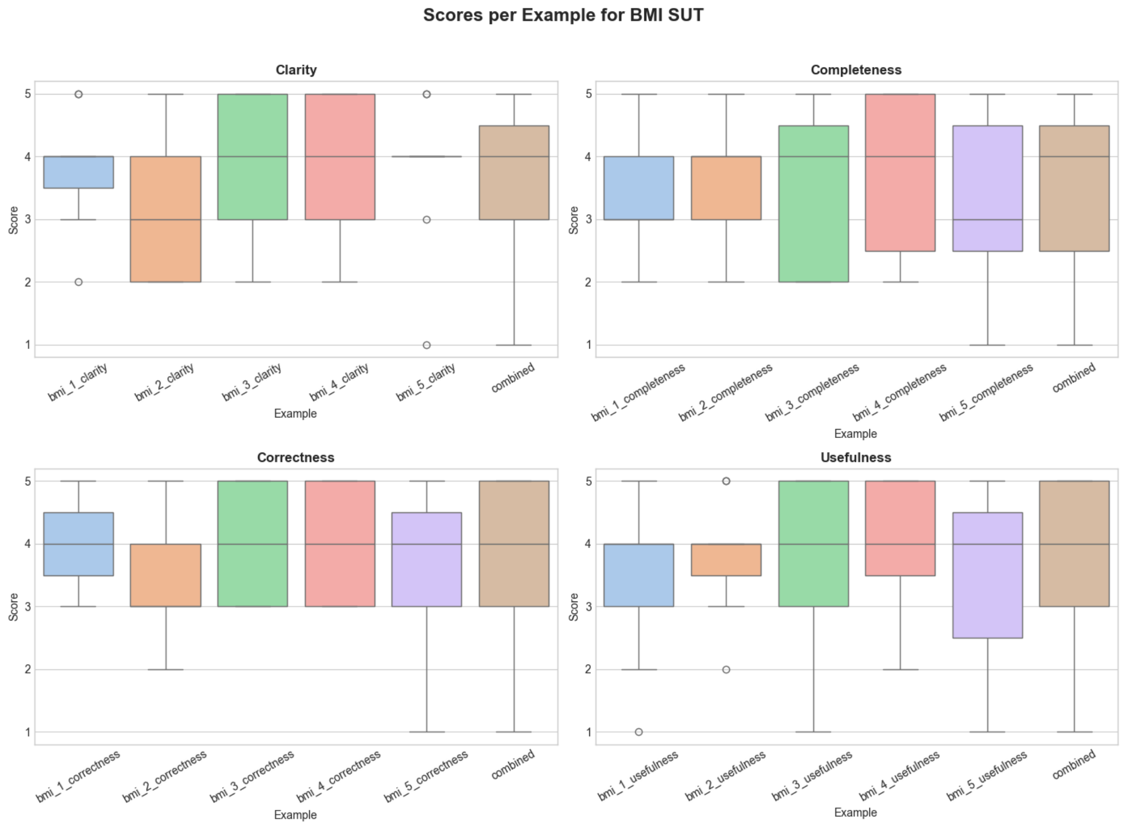
Descriptive Statistics by SUT



	mean	std	min	25%	50%	75%	max
EMAIL	3.9	0.89	1.0	3.0	4.0	5.0	5.0
BYTECNT	3.53	1.22	1.0	3.0	4.0	4.0	5.0
DATE	3.53	1.15	1.0	3.0	4.0	4.0	5.0
BMI	3.71	1.18	1.0	3.0	4.0	5.0	5.0

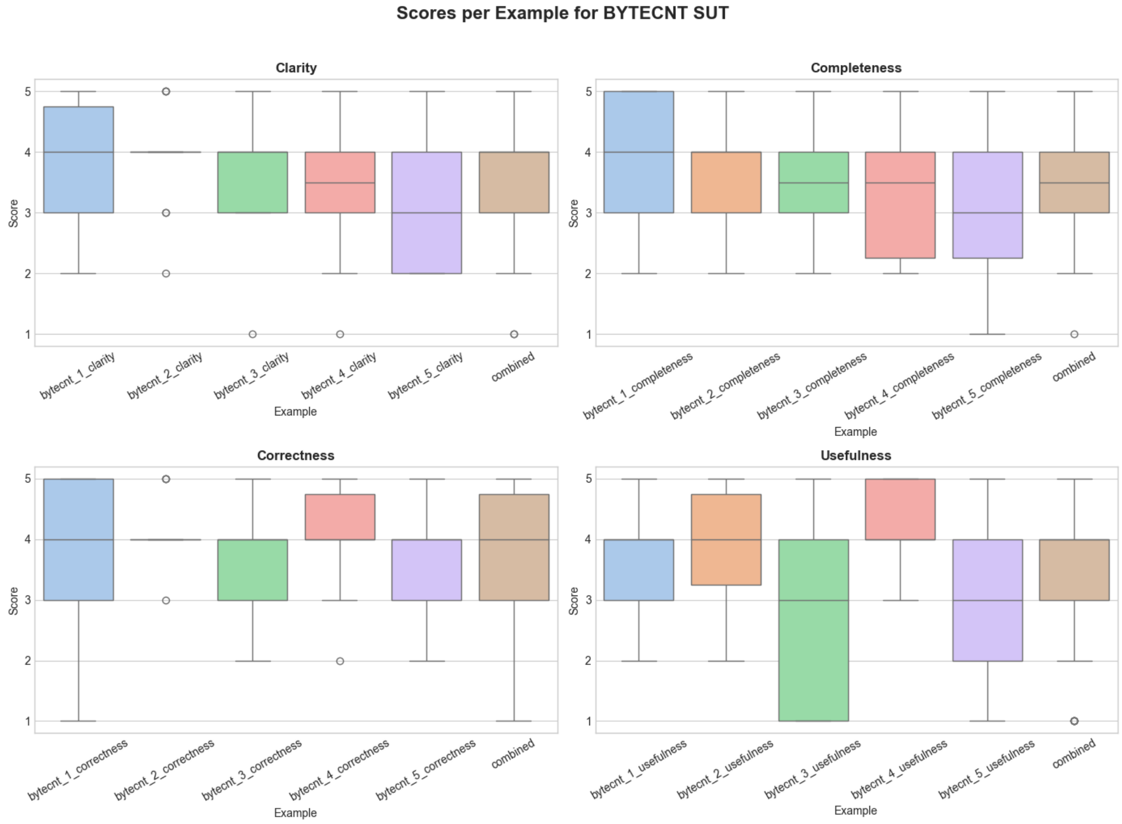
6 Descriptive Statistics: Per SUT

```
[13]: plot_box_per_sut(DF, 'bmi')
```



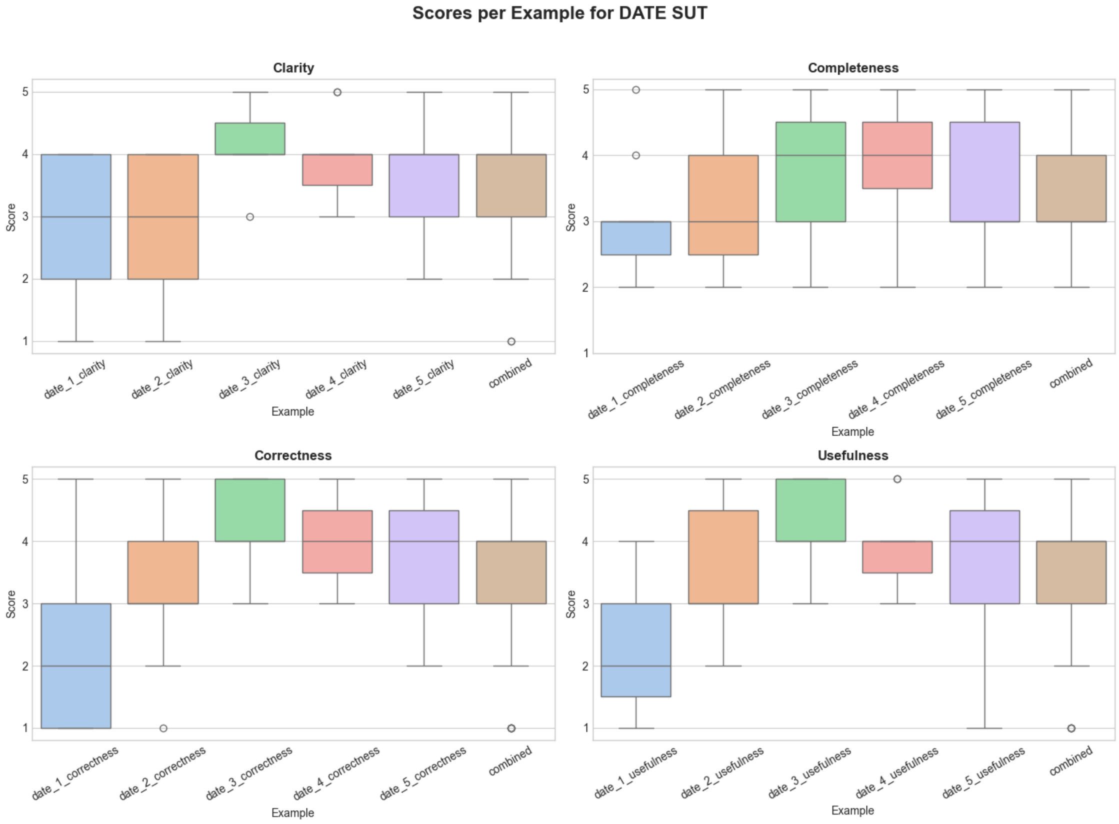
[14]:

```
plot_box_per_sut(DF, 'bytecnt')
```

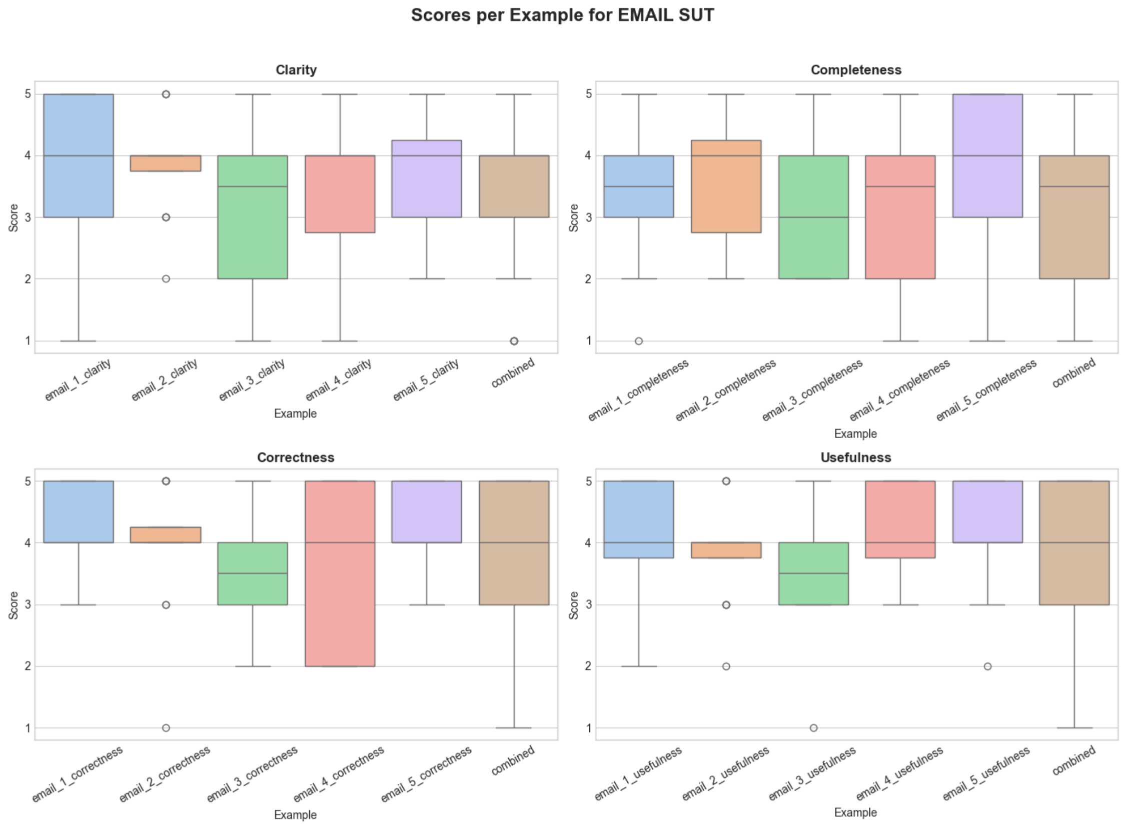


[15]:

```
plot_box_per_sut(DF, 'date')
```



```
[16]: plot_box_per_sut(DF, 'email')
```


Scores per Example for EMAIL SUT
Clarity
Score
5
4
3
2
1
email_1_clarity
email_2_clarity
email_3_clarity
email_4_clarity
email_5_clarity
combined
Example
Completeness
Score
email_1_completeness
email_2_completeness
email_3_completeness
email_4_completeness
email_5_completeness
combined
Example
Correctness
Score
email_1_correctness
email_2_correctness
email_3_correctness
email_4_correctness
email_5_correctness
combined
Example
Usefulness
Score
email_1_usefulness
email_2_usefulness
email_3_usefulness
email_4_usefulness
email_5_usefulness
combined
Example

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```