Evaluation

September 12, 2025

1 Reading in the dataframe

[6]:

```
import pandas as pd
import sklearn
import numpy as np
import matplotlib.pyplot as plt
import krippendorff
from sklearn.metrics import classification_report

sample = "Set-1" #Or Set-2
if sample == "Set-1":
    file = "/home/pc/Uni/MasterThesis/Scripts/
 ↪Corpus_abortion_sample3_chunked_annotated.csv"
    file2 = "/home/pc/Uni/MasterThesis/Scripts/
 ↪Corpus_econ_sample_chunked_annotated.csv"

elif sample == "Set-2":
    file = "/home/pc/Uni/MasterThesis/Scripts/
 ↪Corpus_abort_sample2_chunked_annotated.csv"
    file2 = "/home/pc/Uni/MasterThesis/Scripts/
 ↪Corpus_econ_sample2_chunked_annotated.csv"

Corpus_abortion_sample_chunked_annotated_cleaned = pd.read_csv(file)
Corpus_abortion_sample_chunked_annotated_cleaned =␣
 ↪Corpus_abortion_sample_chunked_annotated_cleaned.fillna(99)
Corpus_abortion_sample_chunked_annotated_cleaned['expert'] =␣
 ↪Corpus_abortion_sample_chunked_annotated_cleaned['expert'].astype(float).
 ↪astype('int')

Corpus_econ_sample_chunked_annotated_cleaned = pd.read_csv(file2)
Corpus_econ_sample_chunked_annotated_cleaned =␣
 ↪Corpus_econ_sample_chunked_annotated_cleaned.fillna(99)
Corpus_econ_sample_chunked_annotated_cleaned['expert'] =␣
 ↪Corpus_econ_sample_chunked_annotated_cleaned['expert'].astype(float).
 ↪astype('int')

del file, file2
```

1.1 Standardizing dataframes

[2]:

```
Column_ordering = ["chunk", "speaker", "file_name", "date", "expert", "student␣
 ↪assistant", "phi:2.7b", "gemma3:4b", "qwen2.5:7b", "mistral:7b", "llama3.2:
 ↪latest", "stablelm2:12b"] #Standard ordering since the first annotation was␣
 ↪less structured
Corpus_econ_sample_chunked_annotated_cleaned =␣
 ↪Corpus_econ_sample_chunked_annotated_cleaned.reindex(columns=Column_ordering)
Corpus_abortion_sample_chunked_annotated_cleaned =␣
 ↪Corpus_abortion_sample_chunked_annotated_cleaned.
 ↪reindex(columns=Column_ordering)

if Corpus_abortion_sample_chunked_annotated_cleaned['student assistant'].
 ↪hasnans:                                    #Exclude student assistant as the␣
 ↪column is only featured in the first set of samples
    Corpus_abortion_sample_chunked_annotated_cleaned.drop(columns=['student␣
 ↪assistant'], inplace=True)
if Corpus_econ_sample_chunked_annotated_cleaned['student assistant'].hasnans:
    Corpus_econ_sample_chunked_annotated_cleaned.drop(columns=['student␣
 ↪assistant'], inplace=True)

if 'Column_ordering' in locals():
    del Column_ordering
```

2 Evaluating metrics

2.1 Speed metrics

[3]:

```
performance_econ_df = pd.read_csv("Model_metrics_econ_sample.csv")
performance_econ_df = performance_econ_df.sort_values(by='Speed',␣
 ↪ascending=False)
performance_econ_df = performance_econ_df.reset_index(drop=True)

performance_econ_df['Speed'] = performance_econ_df['Speed'] /␣
 ↪performance_econ_df['Speed'].max() #Standardizing the speed metric by the␣
 ↪fastest model
print(performance_econ_df[['Model', 'Speed']])
```

```
            Model     Speed
0        phi:2.7b  1.000000
1  llama3.2:latest  0.988633
2       gemma3:4b  0.951229
3      qwen2.5:7b  0.891019
4      mistral:7b  0.839343
5   stablelm2:12b  0.752199
```

[4]:

```
performance_abortion_df = pd.read_csv("Model_metrics_abortion_sample.csv")
performance_abortion_df = performance_abortion_df.sort_values(by='Speed',␣
 ↪ascending=False)
performance_abortion_df = performance_abortion_df.reset_index(drop=True)

performance_abortion_df['Speed'] = performance_abortion_df['Speed'] /␣
 ↪performance_abortion_df['Speed'].max()
print(performance_abortion_df[['Model', 'Speed']])
```

```
             Model     Speed
0         phi:2.7b  1.000000
1  llama3.2:latest  0.976086
2        gemma3:4b  0.942400
3       mistral:7b  0.877817
4        qwen2.5:7b  0.865682
5     stablelm2:12b  0.784476
```

[5]:

```
def model_quality_metrics(sample):
    report = pd.DataFrame(columns=['Model', 'Scores'])

    for model_name in sample.columns[5:]:
        print(f"Model: {model_name}")
        sample[model_name] = sample[model_name].astype(int)
        print(classification_report(sample['expert'], sample[model_name],␣
 ↪labels=[1, 0], target_names=["Topic mentioned", "No mentions"],␣
 ↪zero_division=np.nan))

        report.loc[len(report)] = {'Model': model_name, 'Scores':␣
 ↪classification_report(sample['expert'], sample[model_name],␣
 ↪output_dict=True, zero_division=np.nan)}

    return report

Report_abortion_classification =␣
 ↪model_quality_metrics(Corpus_abortion_sample_chunked_annotated_cleaned)
Report_economic_classification =␣
 ↪model_quality_metrics(Corpus_econ_sample_chunked_annotated_cleaned)
```

```
Model: phi:2.7b
                 precision    recall  f1-score   support

Topic mentioned       0.69      0.64      0.67        64
    No mentions       0.75      0.79      0.77        86

       accuracy                           0.73       150
      macro avg       0.72      0.72      0.72       150
   weighted avg       0.72      0.73      0.72       150
```

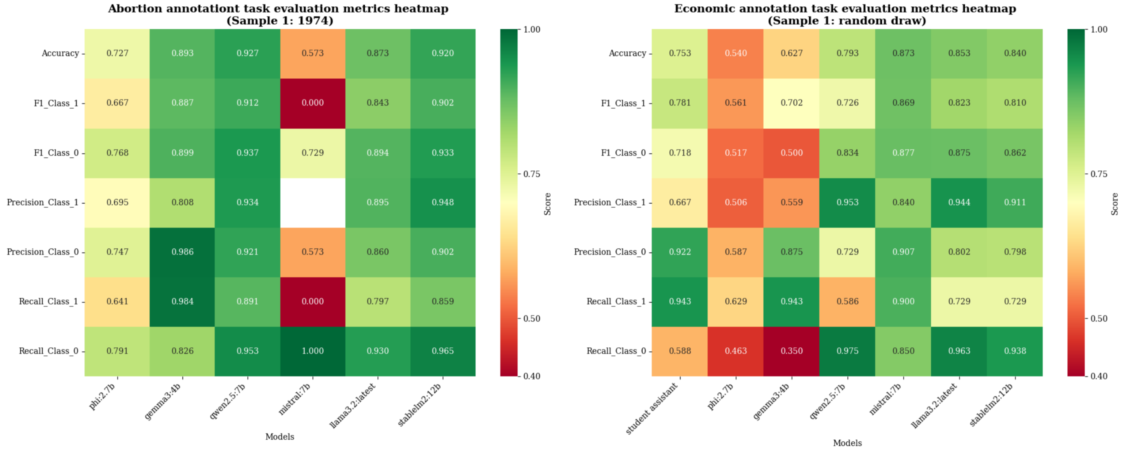
```
Model: gemma3:4b
                 precision    recall  f1-score   support

Topic mentioned       0.81      0.98      0.89        64
    No mentions       0.99      0.83      0.90        86

       accuracy                           0.89       150
      macro avg       0.90      0.90      0.89       150
   weighted avg       0.91      0.89      0.89       150

Model: qwen2.5:7b
                 precision    recall  f1-score   support

Topic mentioned       0.93      0.89      0.91        64
    No mentions       0.92      0.95      0.94        86

       accuracy                           0.93       150
      macro avg       0.93      0.92      0.92       150
   weighted avg       0.93      0.93      0.93       150

Model: mistral:7b
                 precision    recall  f1-score   support

Topic mentioned        nan      0.00      0.00        64
    No mentions       0.57      1.00      0.73        86

       accuracy                           0.57       150
      macro avg       0.57      0.50      0.36       150
   weighted avg       0.57      0.57      0.42       150

Model: llama3.2:latest
                 precision    recall  f1-score   support

Topic mentioned       0.89      0.80      0.84        64
    No mentions       0.86      0.93      0.89        86

       accuracy                           0.87       150
      macro avg       0.88      0.86      0.87       150
   weighted avg       0.87      0.87      0.87       150

Model: stablelm2:12b
                 precision    recall  f1-score   support

Topic mentioned       0.95      0.86      0.90        64
    No mentions       0.90      0.97      0.93        86

       accuracy                           0.92       150
      macro avg       0.93      0.91      0.92       150
   weighted avg       0.92      0.92      0.92       150

Model: student assistant
                 precision    recall  f1-score   support

Topic mentioned       0.67      0.94      0.78        70
    No mentions       0.92      0.59      0.72        80

       accuracy                           0.75       150
      macro avg       0.79      0.77      0.75       150
   weighted avg       0.80      0.75      0.75       150

Model: phi:2.7b
                 precision    recall  f1-score   support

Topic mentioned       0.51      0.63      0.56        70
    No mentions       0.59      0.46      0.52        80

       accuracy                           0.54       150
      macro avg       0.55      0.55      0.54       150
   weighted avg       0.55      0.54      0.54       150

Model: gemma3:4b
                 precision    recall  f1-score   support

Topic mentioned       0.56      0.94      0.70        70
    No mentions       0.88      0.35      0.50        80

       accuracy                           0.63       150
      macro avg       0.72      0.65      0.60       150
   weighted avg       0.73      0.63      0.59       150

Model: qwen2.5:7b
                 precision    recall  f1-score   support

Topic mentioned       0.95      0.59      0.73        70
    No mentions       0.73      0.97      0.83        80

       accuracy                           0.79       150
      macro avg       0.84      0.78      0.78       150
   weighted avg       0.83      0.79      0.78       150

Model: mistral:7b
                 precision    recall  f1-score   support

Topic mentioned       0.84      0.90      0.87        70
    No mentions       0.91      0.85      0.88        80
```

```
       accuracy                           0.87       150
      macro avg       0.87      0.88      0.87       150
   weighted avg       0.88      0.87      0.87       150

Model: llama3.2:latest
                 precision    recall  f1-score   support

Topic mentioned       0.94      0.73      0.82        70
    No mentions       0.80      0.96      0.88        80

       accuracy                           0.85       150
      macro avg       0.87      0.85      0.85       150
   weighted avg       0.87      0.85      0.85       150

Model: stablelm2:12b
                 precision    recall  f1-score   support

Topic mentioned       0.91      0.73      0.81        70
    No mentions       0.80      0.94      0.86        80

       accuracy                           0.84       150
      macro avg       0.85      0.83      0.84       150
   weighted avg       0.85      0.84      0.84       150
```

[ ]:

```
def create_classification_heatmaps(report_abortion, report_economic, sample):
    import seaborn as sns
    import numpy as np

    def extract_metrics(report_df, corpus_name):
        metrics_data = []
        for _, row in report_df.iterrows():
            model_name = row['Model']
            scores = row['Scores']

            metrics_data.append({
                'Model': model_name,
                'Accuracy': scores['accuracy'],
                'F1_Class_1': scores['1']['f1-score'],
                'F1_Class_0': scores['0']['f1-score'],
                'Precision_Class_1': scores['1']['precision'],
                'Recall_Class_1': scores['1']['recall'],
                'Precision_Class_0': scores['0']['precision'],
                'Recall_Class_0': scores['0']['recall']
            })
        return pd.DataFrame(metrics_data)
```

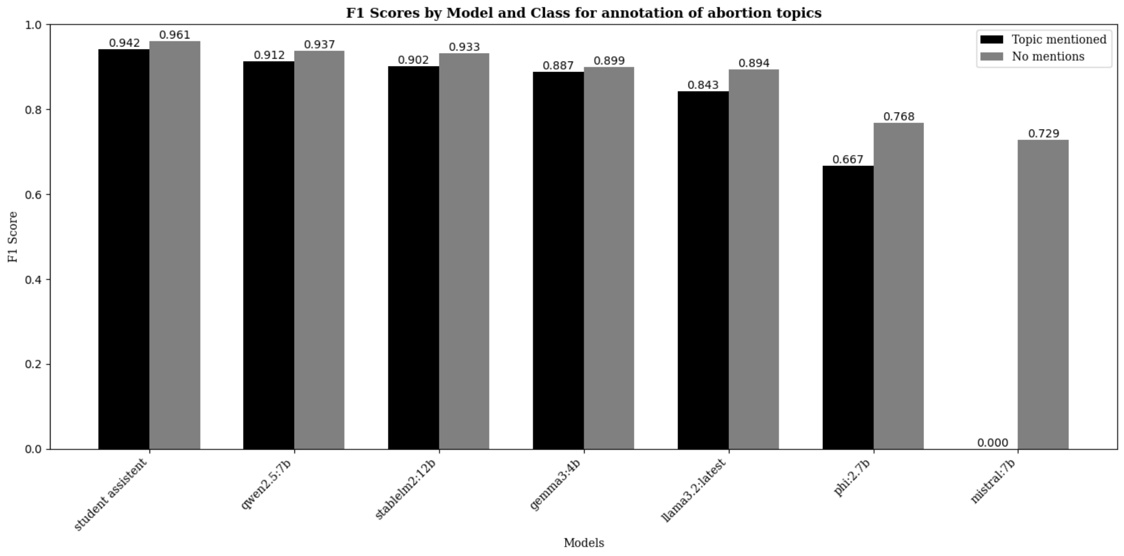
```python
    if sample == "Set-1":
        sample_desc = 'Abortion annotation task evaluation metrics heatmap \n(Sample 1: 1974)'
        sample_desc2 = 'Economic annotation task evaluation metrics heatmap \n(Sample 1: random draw)'
    elif sample == "Set-2":
        sample_desc = 'Abortion annotation task evaluation metrics heatmap \n(Sample 2: 2022)'
        sample_desc2 = 'Economic annotation task evaluation metrics heatmap \n(Sample 2: random draw post 2020)'

    abortion_metrics = extract_metrics(report_abortion, 'Abortion annotation task')
    economic_metrics = extract_metrics(report_economic, 'Economic annotation task')

    fig, axes = plt.subplots(1, 2, figsize=(20, 8))

    # Abortion heatmap
    abortion_heatmap_data = abortion_metrics.set_index('Model')[['Accuracy', 'F1_Class_1', 'F1_Class_0', 'Precision_Class_1', 'Precision_Class_0', 'Recall_Class_1', 'Recall_Class_0']]
    abortion_hm = sns.heatmap(
                abortion_heatmap_data.T, annot=True, cmap='RdYlGn', fmt='.3f',
                cbar_kws={'label': 'Score'}, ax=axes[0],
                annot_kws={"size": 10, "family":"DejaVu Serif"},
                vmin=0.4, vmax=1
                )
    cbar0 = abortion_hm.collections[0].colorbar
    cbar0.ax.yaxis.label.set_fontproperties('DejaVu Serif')
    cbar0.ax.yaxis.label.set_size(10)
    cbar0.set_ticks([0.4, 0.5, 0.75, 1])
    for label in cbar0.ax.get_yticklabels():
        label.set_fontname('DejaVu Serif')
        label.set_fontsize(10)
    axes[0].set_title(sample_desc, fontsize=14, fontweight='bold', family='DejaVu Serif')
    axes[0].set_xlabel('Models', fontsize=10, family='DejaVu Serif')
    axes[0].set_xticklabels(axes[0].get_xticklabels(), rotation=45, horizontalalignment='right', family='DejaVu Serif', fontsize=10)
    axes[0].set_yticklabels(axes[0].get_yticklabels(), rotation=0, family='DejaVu Serif', fontsize=10)

    # Economic heatmap
    economic_heatmap_data = economic_metrics.set_index('Model')[['Accuracy',␣
  ↪'F1_Class_1', 'F1_Class_0', 'Precision_Class_1', 'Precision_Class_0',␣
  ↪'Recall_Class_1', 'Recall_Class_0']]
    economic_hm = sns.heatmap(
        economic_heatmap_data.T, annot=True, cmap='RdYlGn', fmt='.3f',
        cbar_kws={'label': 'Score'}, ax=axes[1],
        annot_kws={"size": 10, "family":"DejaVu Serif"},
        vmin=0.4, vmax=1
    )
    cbar1 = economic_hm.collections[0].colorbar
    cbar1.ax.yaxis.label.set_fontproperties('DejaVu Serif')
    cbar1.ax.yaxis.label.set_size(10)
    cbar1.set_ticks([0.4, 0.5, 0.75, 1])
    for label in cbar1.ax.get_yticklabels():
        label.set_fontname('DejaVu Serif')
        label.set_fontsize(10)
    axes[1].set_title(sample_desc2, fontsize=14, fontweight='bold',␣
  ↪family='DejaVu Serif')
    axes[1].set_xlabel('Models', fontsize=10, family='DejaVu Serif')
    axes[1].set_xticklabels(axes[1].get_xticklabels(), rotation=45,␣
  ↪horizontalalignment='right', family='DejaVu Serif', fontsize=10)
    axes[1].set_yticklabels(axes[1].get_yticklabels(), rotation=0,␣
  ↪family='DejaVu Serif', fontsize=10)

    plt.tight_layout()
    plt.savefig('classification_metrics_heatmaps_sample1.png', dpi=300,␣
  ↪bbox_inches='tight')
    plt.show()

    return abortion_metrics, economic_metrics

abortion_data, economic_data =␣
  ↪create_classification_heatmaps(Report_abortion_classification,␣
  ↪Report_economic_classification, sample)
```



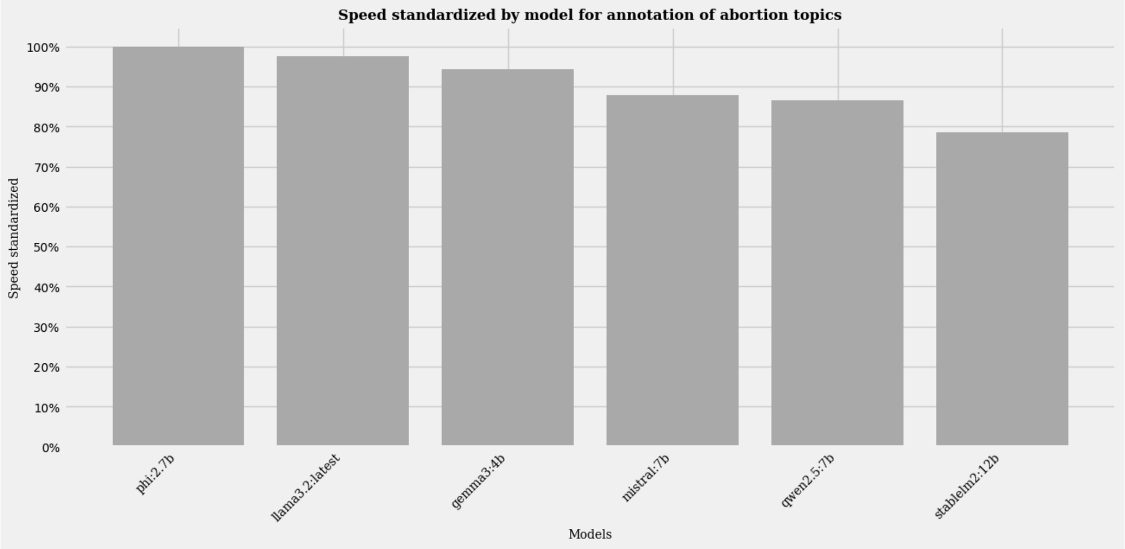
3 Plotting differences

[8]:
```
f1_scored_abort_df = pd.DataFrame(columns=['Model', 'F1_0', 'F1_1'])

for model_name in Corpus_abortion_sample_chunked_annotated_cleaned.columns[5:
 ↪12]:

    Corpus_abortion_sample_chunked_annotated_cleaned[model_name] =␣
 ↪Corpus_abortion_sample_chunked_annotated_cleaned[model_name].astype(int)
    report = classification_report(
        Corpus_abortion_sample_chunked_annotated_cleaned['expert'],
        Corpus_abortion_sample_chunked_annotated_cleaned[model_name],
        output_dict=True,
        zero_division=np.nan
    )

    f1_scored_abort_df.loc[len(f1_scored_abort_df)] = {'Model': model_name,␣
 ↪'F1_0': report['0']['f1-score'], 'F1_1': report['1']['f1-score']}

f1_scored_abort_df = f1_scored_abort_df.sort_values(by='F1_1', ascending=False).
 ↪reset_index().drop(columns=['index'])

plt.figure(figsize=(14, 7))

bar_width = 0.35
x = np.arange(len(f1_scored_abort_df['Model']))

bar1 = plt.bar(x - bar_width/2, f1_scored_abort_df['F1_1'], width=bar_width,␣
 ↪color='black')
```

```
bar2 = plt.bar(x + bar_width/2, f1_scored_abort_df['F1_0'], width=bar_width,␣
 ↪color='grey')

plt.ylabel('F1 Score', fontsize=10, family='DejaVu Serif')
plt.bar_label(bar1, fontsize=10, fmt='%.3f')
plt.bar_label(bar2, fontsize=10, fmt='%.3f')
plt.title('F1 Scores by Model and Class for annotation of abortion topics',␣
 ↪fontsize=12, fontweight='bold', family='DejaVu Serif')
plt.xlabel('Models', fontsize=10, family='DejaVu Serif')
plt.xticks(x, f1_scored_abort_df['Model'], rotation=45, ha='right',␣
 ↪fontname='DejaVu Serif', fontsize=10)
plt.legend([bar1, bar2], ['Topic mentioned', 'No mentions'], prop={'family':␣
 ↪'DejaVu Serif', 'size':10})
plt.yticks(fontsize=10)
plt.ylim(0, 1)

plt.tight_layout()
plt.style.use('fivethirtyeight')
plt.show()

del bar1, bar2, x, bar_width, report, model_name #Saving RAM saves lives.
```



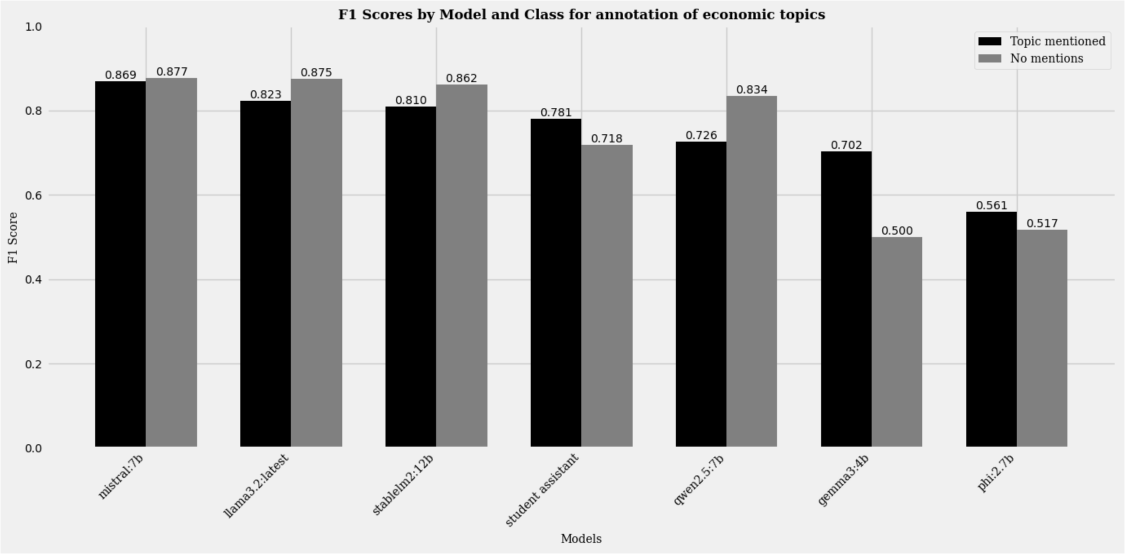
[9]:
```
plt.figure(figsize=(14, 6))
plt.bar(performance_abortion_df['Model'],␣
 ↪height=performance_abortion_df['Speed'], color = 'darkgrey')
plt.xlabel('Models', fontsize=10, family='DejaVu Serif')
plt.ylabel('Speed standardized', fontsize=10, family='DejaVu Serif')
plt.title('Speed standardized by model for annotation of abortion topics', fontsize=12, fontweight='bold', family='DejaVu Serif')
plt.xticks(rotation=45, ha='right', fontname='DejaVu Serif', fontsize = 10)
plt.yticks(np.arange(0, 1.1, 0.1), ['0%', '10%', '20%', '30%', '40%', '50%', '60%', '70%', '80%', '90%', '100%'], fontsize=10)
plt.style.use('fivethirtyeight')
plt.savefig('abortion_annotation_speed.png', dpi=300, bbox_inches='tight')
plt.show()
```



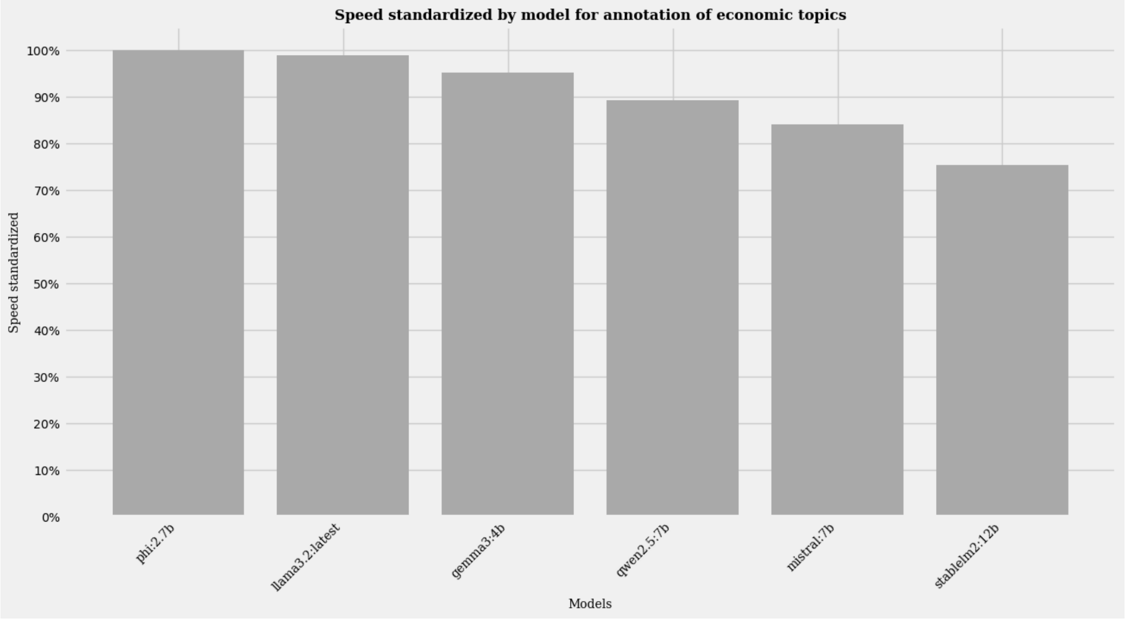
[10]:

```
f1_scored_econ_df = pd.DataFrame(columns=['Model', 'F1_0', 'F1_1'])

for model_name in Corpus_econ_sample_chunked_annotated_cleaned.columns[5:12]:
    report = classification_report(
        Corpus_econ_sample_chunked_annotated_cleaned['expert'],
        Corpus_econ_sample_chunked_annotated_cleaned[model_name],
        output_dict=True
    )

    f1_scored_econ_df.loc[len(f1_scored_econ_df)] = {'Model': model_name, 'F1_0': report['0']['f1-score'], 'F1_1': report['1']['f1-score']}

f1_scored_econ_df = f1_scored_econ_df.sort_values(by='F1_1', ascending=False).reset_index().drop(columns=['index'])

plt.figure(figsize=(14, 7))

bar_width = 0.35
x = np.arange(len(f1_scored_econ_df['Model']))
```

```python
bar1 = plt.bar(x - bar_width/2, f1_scored_econ_df['F1_1'], width=bar_width, color='black')
bar2 = plt.bar(x + bar_width/2, f1_scored_econ_df['F1_0'], width=bar_width, color='grey')

plt.ylabel('F1 Score', fontsize=10, family='DejaVu Serif')
plt.bar_label(bar1, fontsize=10, fmt='%.3f')
plt.bar_label(bar2, fontsize=10, fmt='%.3f')
plt.title('F1 Scores by Model and Class for annotation of economic topics', fontsize=12, fontweight='bold', family='DejaVu Serif')
plt.xlabel('Models', fontsize=10, family='DejaVu Serif')
plt.xticks(x, f1_scored_econ_df['Model'], rotation=45, ha='right', fontname='DejaVu Serif', fontsize=10)
plt.legend([bar1, bar2], ['Topic mentioned', 'No mentions'], prop={'family': 'DejaVu Serif', 'size':10})
plt.yticks(fontsize=10)
plt.ylim(0, 1)

plt.tight_layout()
plt.style.use('fivethirtyeight')
plt.show()
del bar1, bar2, x, bar_width, report, model_name
```



[11]:
```python
plt.figure(figsize=(14, 7))
plt.bar(performance_econ_df['Model'], height=performance_econ_df['Speed'], color = 'darkgrey')
plt.xlabel('Models', fontsize=10, family='DejaVu Serif')
plt.ylabel('Speed standardized', fontsize=10, family='DejaVu Serif')
plt.yticks(np.arange(0, 1.1, 0.1), ['0%', '10%', '20%', '30%', '40%', '50%',␣
 ↪'60%', '70%', '80%', '90%', '100%'], fontsize=10)
plt.title('Speed standardized by model for annotation of economic topics',␣
 ↪fontsize=12, fontweight='bold', family='DejaVu Serif')
plt.xticks(rotation=45, ha='right', fontname='DejaVu Serif', fontsize=10)
plt.style.use('fivethirtyeight')
plt.savefig('economic_annotation_speed.png', dpi=300, bbox_inches='tight')
plt.show()
```



3.1 Excluding outlier models

[12]:
```
Corpus_abortion_selected =␣
 ↪Corpus_abortion_sample_chunked_annotated_cleaned[["qwen2.5:7b","gemma3:4b",␣
 ↪"llama3.2:latest", "stablelm2:12b","expert"]] #removing Phi and Mistral due␣
 ↪to especially low performance
Corpus_econ_selected = Corpus_econ_sample_chunked_annotated_cleaned[["gemma3:
 ↪4b", "llama3.2:latest", "stablelm2:12b", "mistral:7b", "expert"]] #removing␣
 ↪Phi and Mistral due to especially low performance
```

3.2 Calculating intermodel-aggreement

[13]:
```
def krippendorff_alpha(corpus1, corpus2):
    annotation_cols_abort = corpus1.columns[~corpus1.columns.isin(['mean_agreement', 'agreement', 'expert', 'student assistent', 'chunk', 'speaker', 'date', 'file_name','processing_type'])]
    annotation_cols_econ = corpus2.columns[~corpus2.columns.isin(['mean_agreement', 'agreement', 'expert', 'student assistent', 'chunk', 'speaker', 'date', 'file_name','processing_type'])]

    data_for_alpha_abort = corpus1[annotation_cols_abort].T.values
    data_for_alpha_econ = corpus2[annotation_cols_econ].T.values

    alpha_abort = krippendorff.alpha(reliability_data=data_for_alpha_abort, level_of_measurement='nominal')
    print(f"Krippendorff's Alpha (Abortion Corpus): {alpha_abort:.3f}")

    alpha_econ = krippendorff.alpha(reliability_data=data_for_alpha_econ, level_of_measurement='nominal')
    print(f"Krippendorff's Alpha (Economic Corpus): {alpha_econ:.3f}")

def agreement_score(corpus1, corpus2, sigma=0.166):
    cols_to_average1 = corpus1.columns[~corpus1.columns.isin(['mean_agreement', 'agreement', 'expert', 'student assistent', 'chunk', 'speaker', 'date', 'file_name','processing_type'])]
    cols_to_average2 = corpus2.columns[~corpus2.columns.isin(['mean_agreement', 'agreement', 'expert', 'student assistent', 'chunk', 'speaker', 'date', 'file_name','processing_type'])]

    corpus1['mean_agreement'] = corpus1[cols_to_average1].mean(axis=1)
    corpus2['mean_agreement'] = corpus2[cols_to_average2].mean(axis=1)

    def inverted_gaussian(x, mu=0.5, sigma=sigma):
            return 1 - np.exp(-((x - mu) ** 2) / (2 * sigma ** 2))

    corpus1['agreement'] = corpus1['mean_agreement'].apply(inverted_gaussian)
    corpus2['agreement'] = corpus2['mean_agreement'].apply(inverted_gaussian)


    return corpus1, corpus2

if __name__ == "__main__":
    print("Krippendorff for selected models:\n")
    krippendorff_alpha(Corpus_abortion_selected, Corpus_econ_selected)
    print("\nKrippendorff for all models:\n")
    krippendorff_alpha(Corpus_abortion_sample_chunked_annotated_cleaned, Corpus_econ_sample_chunked_annotated_cleaned)
```

```python
agreement_score(Corpus_abortion_sample_chunked_annotated_cleaned, Corpus_econ_sample_chunked_annotated_cleaned)
agreement_score(Corpus_abortion_selected, Corpus_econ_selected)

print(f'\nAgreement metrics for the abortion sample \n{Corpus_abortion_sample_chunked_annotated_cleaned['agreement'].describe()}\n')
print(f'Agreement metrics for the econ sample \n{Corpus_econ_sample_chunked_annotated_cleaned['agreement'].describe()}\n')

print("Agreement score after removing Phi and Mistral \n")
print(f'Agreement metrics for the abortion sample w.o. Phi and Mistral\n {Corpus_abortion_selected['agreement'].describe()}\n')
print(f'Agreement metrics for the econ sample w.o. Phi {Corpus_econ_selected['agreement'].describe()}')
```

```
Krippendorff for selected models:

Krippendorff's Alpha (Abortion Corpus): 0.750
Krippendorff's Alpha (Economic Corpus): 0.399

Krippendorff for all models:

Krippendorff's Alpha (Abortion Corpus): 0.400
Krippendorff's Alpha (Economic Corpus): 0.311

Agreement metrics for the abortion sample
count    150.000000
mean       0.758623
std        0.301154
min        0.000000
25%        0.395905
50%        0.866826
75%        0.989286
max        0.989286
Name: agreement, dtype: float64

Agreement metrics for the econ sample
count    150.000000
mean       0.624645
std        0.365436
min        0.088420
25%        0.088420
50%        0.901174
75%        0.901174
max        0.989286
Name: agreement, dtype: float64
```

```
Agreement score after removing Phi and Mistral

Agreement metrics for the abortion sample w.o. Phi and Mistral
 count    150.000000
mean       0.871123
std        0.269217
min        0.000000
25%        0.989286
50%        0.989286
75%        0.989286
max        0.989286
Name: agreement, dtype: float64

Agreement metrics for the econ sample w.o. Phi count    150.000000
mean       0.752189
std        0.276020
min        0.000000
25%        0.678274
50%        0.678274
75%        0.989286
max        0.989286
Name: agreement, dtype: float64
```

```
/tmp/ipykernel_74902/2519639514.py:19: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  corpus1['mean_agreement'] = corpus1[cols_to_average1].mean(axis=1)
/tmp/ipykernel_74902/2519639514.py:20: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  corpus2['mean_agreement'] = corpus2[cols_to_average2].mean(axis=1)
/tmp/ipykernel_74902/2519639514.py:25: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  corpus1['agreement'] = corpus1['mean_agreement'].apply(inverted_gaussian)
/tmp/ipykernel_74902/2519639514.py:26: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  corpus2['agreement'] = corpus2['mean_agreement'].apply(inverted_gaussian)
```

3.3 Evaluating problematic chunks

[14]:

```
print(f'Mean length for problematic chunks (abortion):␣
  ↪{Corpus_abortion_sample_chunked_annotated_cleaned[Corpus_abortion_sample_chunked_annotated_
  ↪<= 0.5]['chunk'].apply(len).mean().round()} \n')
print(f'Mean length for problematic chunks (economic):␣
  ↪{Corpus_econ_sample_chunked_annotated_cleaned[Corpus_econ_sample_chunked_annotated_cleaned[
  ↪<= 0.5]['chunk'].apply(len).mean().round()} \n')

print(f'Mean length non-problematic chunks (abortion):␣
  ↪{Corpus_abortion_sample_chunked_annotated_cleaned[Corpus_abortion_sample_chunked_annotated_
  ↪> 0.5]['chunk'].apply(len).mean().round()} \n')
print(f'Mean length non-problematic chunks (economic):␣
  ↪{Corpus_econ_sample_chunked_annotated_cleaned[Corpus_econ_sample_chunked_annotated_cleaned[
  ↪> 0.5]['chunk'].apply(len).mean().round()} \n')
```

```
Mean length for problematic chunks (abortion): 6425.0

Mean length for problematic chunks (economic): 7117.0

Mean length non-problematic chunks (abortion): 2945.0

Mean length non-problematic chunks (economic): 4456.0
```