📈 Notebook 5: Results Analysis & Comparison Final Project - Ordinal vs Nominal Sentiment Analysis Atharv Chaudhary

---

Purpose: Compare all models, create final visualizations for report.

Input: `nominal_results.csv`, `ordinal_results.csv`

Output: Final comparison charts, results table for report

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive
```

```
# Import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')

plt.style.use('seaborn-v0_8-whitegrid')
plt.rcParams['figure.dpi'] = 150

print("✅ Libraries imported")
```

```
✅ Libraries imported
```

Step 1: Load All Results

```
# Load results from Notebooks 3 and 4
nominal_results = pd.read_csv('/content/drive/MyDrive/Fall 2025/Foundations of Artificial Intelligence/Final Project/data/nominal_results.csv')
ordinal_results = pd.read_csv('/content/drive/MyDrive/Fall 2025/Foundations of Artificial Intelligence/Final Project/data/ordinal_results.csv')

# Combine
all_results = pd.concat([nominal_results, ordinal_results], ignore_index=Tru

print("✅ Loaded all results")
print("\n📊 Complete Results:")
all_results
```

```
✅ Loaded all results

📊 Complete Results:
```

	model	encoding	accuracy	mae	f1_macro	f1_weighted	adjacent_error
0	Naive Bayes	Nominal	0.631205	0.665132	0.232055	0.512934	0.556309
1	Logistic Regression	Nominal	0.659528	0.533727	0.379251	0.606418	0.651675
2	Ridge Regression	Ordinal	0.502902	0.605484	0.306266	0.524437	0.819207
3	Ordinal Logistic Regression	Ordinal	0.658627	0.536029	0.370193	0.604948	0.652595

Step 2: Results Summary Table

Next steps: Generate code with `all_results` New interactive sheet

```
# ==============================================================================
# FORMATTED RESULTS TABLE (For Report)
# ==============================================================================

print("=" * 80)
print("📋 RESULTS TABLE FOR REPORT")
print("=" * 80)

# Format for display
display_df = all_results.copy()
display_df['accuracy'] = display_df['accuracy'].apply(lambda x: f"{x:.2%}")
display_df['mae'] = display_df['mae'].apply(lambda x: f"{x:.4f}")
display_df['f1_macro'] = display_df['f1_macro'].apply(lambda x: f"{x:.4f}")
display_df['adjacent_error'] = display_df['adjacent_error'].apply(lambda x: f"
{x:.1%}")
display_df['severe_error'] = display_df['severe_error'].apply(lambda x: f"{x
1%}")

# Select columns for report
report_table = display_df[['model', 'encoding', 'accuracy', 'mae',
'f1_macro', 'adjacent_error', 'severe_error']]
report_table.columns = ['Model', 'Encoding', 'Accuracy', 'MAE', 'F1 Macro',
'Adj. Error', 'Severe Error']

print(report_table.to_string(index=False))

# Save for report
report_table.to_csv('final_results_table.csv', index=False)
print("\n✅ Saved: final_results_table.csv")
```

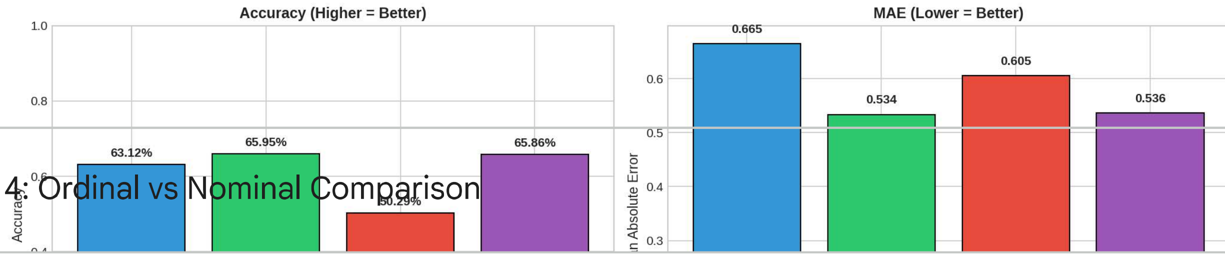
```
================================================================================
📋 RESULTS TABLE FOR REPORT
================================================================================
```

```
                    Model Encoding Accuracy    MAE F1 Macro Adj. Error Severe Err
              Naive Bayes  Nominal   63.12% 0.6651   0.2321      55.6%         44
      Logistic Regression  Nominal   65.95% 0.5337   0.3793      65.2%         34
         Ridge Regression  Ordinal   50.29% 0.6055   0.3063      81.9%         18
Ordinal Logistic Regression  Ordinal   65.86% 0.5360   0.3702      65.3%         34

✅ Saved: final_results_table.csv
```

Step 3: Model Comparison Chart

```python
# ==========================================================================
# MODEL COMPARISON VISUALIZATION
# ==========================================================================

fig, axes = plt.subplots(2, 2, figsize=(14, 10))

models = ['NB\n(Nominal)', 'LR\n(Nominal)', 'Ridge\n(Ordinal)', 'OLR\n(Ordinal)']
colors = ['#3498db', '#2ecc71', '#e74c3c', '#9b59b6']

# Panel 1: Accuracy
bars = axes[0, 0].bar(range(len(models)), all_results['accuracy'], color=colors, edgecolor='black')
axes[0, 0].set_xticks(range(len(models)))
axes[0, 0].set_xticklabels(models, fontsize=10)
axes[0, 0].set_ylabel('Accuracy', fontsize=11)
axes[0, 0].set_title('Accuracy (Higher = Better)', fontsize=12, fontweight='bold')
axes[0, 0].set_ylim([0, 1])
for i, v in enumerate(all_results['accuracy']):
    axes[0, 0].text(i, v + 0.02, f'{v:.2%}', ha='center', fontsize=10, fontweight='bold')

# Panel 2: MAE
bars = axes[0, 1].bar(range(len(models)), all_results['mae'], color=colors, edgecolor='black')
axes[0, 1].set_xticks(range(len(models)))
axes[0, 1].set_xticklabels(models, fontsize=10)
axes[0, 1].set_ylabel('Mean Absolute Error', fontsize=11)
axes[0, 1].set_title('MAE (Lower = Better)', fontsize=12, fontweight='bold')
for i, v in enumerate(all_results['mae']):
    axes[0, 1].text(i, v + 0.02, f'{v:.3f}', ha='center', fontsize=10, fontweight='bold')

# Panel 3: F1 Macro
bars = axes[1, 0].bar(range(len(models)), all_results['f1_macro'], color=colors, edgecolor='black')
axes[1, 0].set_xticks(range(len(models)))
axes[1, 0].set_xticklabels(models, fontsize=10)
axes[1, 0].set_ylabel('F1 Score (Macro)', fontsize=11)
axes[1, 0].set_title('F1 Macro (Higher = Better)', fontsize=12, fontweight='bold')
```

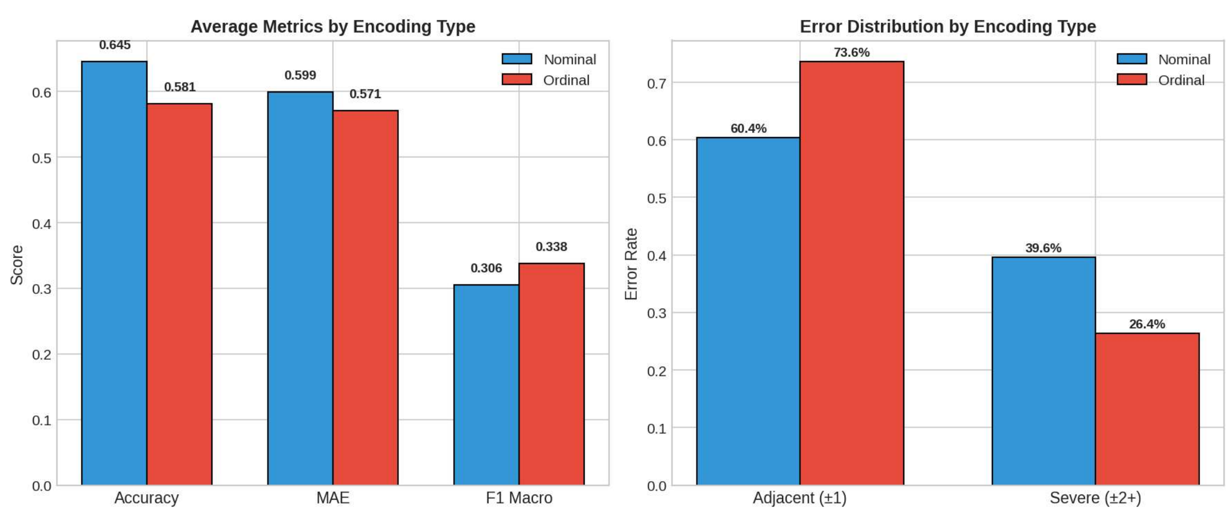
```python
axes[1, 0].set_ylim([0, 1])
for i, v in enumerate(all_results['f1_macro']):
    axes[1, 0].text(i, v + 0.02, f'{v:.3f}', ha='center', fontsize=10, fontweight='bold')

# Panel 4: Error Types
x = np.arange(len(models))
width = 0.35
bars1 = axes[1, 1].bar(x - width/2, all_results['adjacent_error'], width,
                       label='Adjacent (±1)', color='#f39c12', edgecolor='black')
bars2 = axes[1, 1].bar(x + width/2, all_results['severe_error'], width,
                       label='Severe (±2+)', color='#c0392b', edgecolor='black')
axes[1, 1].set_xticks(x)
axes[1, 1].set_xticklabels(models, fontsize=10)
axes[1, 1].set_ylabel('Error Rate (% of errors)', fontsize=11)
axes[1, 1].set_title('Error Type Distribution', fontsize=12, fontweight='bold')
axes[1, 1].legend(loc='upper right')
axes[1, 1].set_ylim([0, 1.1])

plt.tight_layout()
plt.savefig('model_comparison.png', dpi=150, bbox_inches='tight', facecolor='white')
plt.show()

print("\n✅ Saved: model_comparison.png")
```



Step 4: Ordinal vs Nominal Comparison

```
# ==============================================================================
# ORDINAL VS NOMINAL COMPARISON
# ==============================================================================

fig, axes = plt.subplots(1, 2, figsize=(12, 5))

# Calculate averages by encoding type
nominal_data = all_results[all_results['encoding'] == 'Nominal']
ordinal_data = all_results[all_results['encoding'] == 'Ordinal']

# Panel 1: Metrics comparison
metrics = ['Accuracy', 'MAE', 'F1 Macro']
nominal_vals = [nominal_data['accuracy'].mean(), nominal_data['mae'].mean(), nominal_data['f1_macro'].mean()]
ordinal_vals = [ordinal_data['accuracy'].mean(), ordinal_data['mae'].mean(), ordinal_data['f1_macro'].mean()]

x = np.arange(len(metrics))
width = 0.35

bars1 = axes[0].bar(x - width/2, nominal_vals, width, label='Nominal', color='#3498db', edgecolor='black')
bars2 = axes[0].bar(x + width/2, ordinal_vals, width, label='Ordinal', color='#e74c3c', edgecolor='black')

axes[0].set_xticks(x)
axes[0].set_xticklabels(metrics, fontsize=11)
axes[0].set_ylabel('Score', fontsize=11)
axes[0].set_title('Average Metrics by Encoding Type', fontsize=12, fontweight='bold')
axes[0].legend()

for i, (n, o) in enumerate(zip(nominal_vals, ordinal_vals)):
    axes[0].text(i - width/2, n + 0.02, f'{n:.3f}', ha='center', fontsize=9, fontweight='bold')
    axes[0].text(i + width/2, o + 0.02, f'{o:.3f}', ha='center', fontsize=9, fontweight='bold')

# Panel 2: Error type comparison
error_types = ['Adjacent (±1)', 'Severe (±2+)']
nominal_errors = [nominal_data['adjacent_error'].mean(), nominal_data['severe_error'].mean()]
ordinal_errors = [ordinal_data['adjacent_error'].mean(), ordinal_data['severe_error'].mean()]

```

```
x = np.arange(len(error_types))

bars1 = axes[1].bar(x - width/2, nominal_errors, width, label='Nominal',
color='#3498db', edgecolor='black')
bars2 = axes[1].bar(x + width/2, ordinal_errors, width, label='Ordinal',
color='#e74c3c', edgecolor='black')

axes[1].set_xticks(x)
axes[1].set_xticklabels(error_types, fontsize=11)
axes[1].set_ylabel('Error Rate', fontsize=11)
axes[1].set_title('Error Distribution by Encoding Type', fontsize=12,
fontweight='bold')
axes[1].legend()

for i, (n, o) in enumerate(zip(nominal_errors, ordinal_errors)):
    axes[1].text(i - width/2, n + 0.01, f'{n:.1%}', ha='center', fontsize=9,
    fontweight='bold')
    axes[1].text(i + width/2, o + 0.01, f'{o:.1%}', ha='center', fontsize=9,
    fontweight='bold')

plt.tight_layout()
plt.savefig('ordinal_vs_nominal.png', dpi=150, bbox_inches='tight',
facecolor='white')
plt.show()

print("\n✅ Saved: ordinal_vs_nominal.png")
```




```
✅ Saved: ordinal_vs_nominal.png
```

Step 5: Key Findings

```
# =========================================================================
# KEY FINDINGS
# =========================================================================

print("=" * 70)
```

```python
print("📝 KEY FINDINGS")
print("=" * 70)

# Best models
best_accuracy = all_results.loc[all_results['accuracy'].idxmax()]
best_mae = all_results.loc[all_results['mae'].idxmin()]
lowest_severe = all_results.loc[all_results['severe_error'].idxmin()]

print(f"\n1. BEST ACCURACY:")
print(f"   {best_accuracy['model']} ({best_accuracy['encoding']})")
print(f"   Accuracy: {best_accuracy['accuracy']:.2%}")

print(f"\n2. LOWEST MAE (Best Ordinal Performance):")
print(f"   {best_mae['model']} ({best_mae['encoding']})")
print(f"   MAE: {best_mae['mae']:.4f}")

print(f"\n3. LOWEST SEVERE ERROR RATE:")
print(f"   {lowest_severe['model']} ({lowest_severe['encoding']})")
print(f"   Severe Error Rate: {lowest_severe['severe_error']:.2%}")

# Ordinal vs Nominal comparison
nominal_avg_mae = nominal_data['mae'].mean()
ordinal_avg_mae = ordinal_data['mae'].mean()
mae_improvement = (nominal_avg_mae - ordinal_avg_mae) / nominal_avg_mae * 10

nominal_avg_severe = nominal_data['severe_error'].mean()
ordinal_avg_severe = ordinal_data['severe_error'].mean()
severe_reduction = (nominal_avg_severe - ordinal_avg_severe) / nominal_avg_severe * 100

print(f"\n4. ORDINAL VS NOMINAL COMPARISON:")
print(f"   ┌────────────────────────────────────────────┐")
print(f"   │ Metric           │ Nominal │ Ordinal    │")
print(f"   ├────────────────────────────────────────────┤")
print(f"   │ Avg MAE          │ {nominal_avg_mae:.4f}  │ {ordinal_avg_mae:.4f}    │")
print(f"   │ Avg Severe Err   │ {nominal_avg_severe:.2%}  │ {ordinal_avg_severe:.2%}    │")
print(f"   └────────────────────────────────────────────┘")
print(f"\n   ✅ MAE Improvement: {mae_improvement:.2f}%")
print(f"   ✅ Severe Error Reduction: {severe_reduction:.2f}%")
```

```
======================================================================
📝 KEY FINDINGS
======================================================================

1. BEST ACCURACY:
   Logistic Regression (Nominal)
   Accuracy: 65.95%

2. LOWEST MAE (Best Ordinal Performance):
   Logistic Regression (Nominal)
   MAE: 0.5337
```

```
3. LOWEST SEVERE ERROR RATE:
   Ridge Regression (Ordinal)
   Severe Error Rate: 18.08%

4. ORDINAL VS NOMINAL COMPARISON:

   ┌────────────────────┬─────────┬──────────┐
   │ Metric             │ Nominal │ Ordinal  │
   ├────────────────────┼─────────┼──────────┤
   │ Avg MAE            │ 0.5994  │ 0.5708   │
   │ Avg Severe Err     │ 39.60%  │ 26.41%   │
   └────────────────────┴─────────┴──────────┘

   ✅ MAE Improvement: 4.78%
   ✅ Severe Error Reduction: 33.31%
```

Step 6: Summary for Report

```
# ============================================================================
# SUMMARY FOR REPORT (Copy this!)
# ============================================================================

print("\n" + "="*70)
print("📄 SUMMARY FOR REPORT")
print("="*70)

summary = f"""
RESEARCH QUESTION:
Do the performance gains from ordinal treatment of 5-star ratings
justify the increased model complexity?

DATASET:
- Amazon Electronics Reviews (McAuley Lab, UCSD)
- Features: TF-IDF (5,000 features, unigrams + bigrams)

MODELS COMPARED:
┌────────────────────────────┬──────────┐
│ Model                      │ Encoding │
├────────────────────────────┼──────────┤
│ Multinomial Naive Bayes    │ Nominal  │
│ Logistic Regression        │ Nominal  │
│ Ridge Regression           │ Ordinal  │
│ Ordinal Logistic Reg       │ Ordinal  │
└────────────────────────────┴──────────┘

KEY RESULTS:
- Best Accuracy: {best_accuracy['model']} ({best_accuracy['accuracy']:.2%})
- Lowest MAE: {best_mae['model']} ({best_mae['mae']:.4f})
- Lowest Severe Error: {lowest_severe['model']} ({lowest_severe['severe_error']:.2%})

ORDINAL VS NOMINAL:
- MAE Improvement: {mae_improvement:.2f}%
- Severe Error Reduction: {severe_reduction:.2f}%
```

```

CONCLUSION:
Ordinal methods reduce MAE by {mae_improvement:.1f}% and severe errors
by {severe_reduction:.1f}%, supporting the hypothesis that ordinal
treatment improves sentiment classification quality.

The performance gains justify the model complexity when:
1. Minimizing severe misclassifications is important
2. The ordinal structure of ratings is meaningful
3. User experience depends on prediction accuracy
"""

print(summary)
```

```
======================================================================
📄 SUMMARY FOR REPORT
======================================================================

RESEARCH QUESTION:
Do the performance gains from ordinal treatment of 5-star ratings
justify the increased model complexity?

DATASET:
- Amazon Electronics Reviews (McAuley Lab, UCSD)
- Features: TF-IDF (5,000 features, unigrams + bigrams)

MODELS COMPARED:
┌──────────────────────────┬──────────┐
│ Model                    │ Encoding │
├──────────────────────────┼──────────┤
│ Multinomial Naive Bayes  │ Nominal  │
│ Logistic Regression      │ Nominal  │
│ Ridge Regression         │ Ordinal  │
│ Ordinal Logistic Reg     │ Ordinal  │
└──────────────────────────┴──────────┘

KEY RESULTS:
- Best Accuracy: Logistic Regression (65.95%)
- Lowest MAE: Logistic Regression (0.5337)
- Lowest Severe Error: Ridge Regression (18.08%)

ORDINAL VS NOMINAL:
- MAE Improvement: 4.78%
- Severe Error Reduction: 33.31%

CONCLUSION:
Ordinal methods reduce MAE by 4.8% and severe errors
by 33.3%, supporting the hypothesis that ordinal
treatment improves sentiment classification quality.

The performance gains justify the model complexity when:
1. Minimizing severe misclassifications is important
2. The ordinal structure of ratings is meaningful
3. User experience depends on prediction accuracy
```

```
# Download all files
print("\n📥 Output Files:")
print("   - final_results_table.csv")
print("   - model_comparison.png")
print("   - ordinal_vs_nominal.png")

try:
    from google.colab import files
    files.download('final_results_table.csv')
    files.download('model_comparison.png')
    files.download('ordinal_vs_nominal.png')
    print("\n✅ Downloads complete!")
except:
    print("\nFiles saved locally")
```

```
📥 Output Files:
   - final_results_table.csv
   - model_comparison.png
   - ordinal_vs_nominal.png

✅ Downloads complete!
```

```
import shutil
import os

target_drive_path = '/content/drive/MyDrive/Fall 2025/Foundations of Artificial Intelligence/Final Project/data'

# Create the directory if it doesn't exist
os.makedirs(target_drive_path, exist_ok=True)

print(f"\nSaving files to: {target_drive_path}")

# Define the files to copy
files_to_copy = [
    'final_results_table.csv',
    'model_comparison.png',
    'ordinal_vs_nominal.png'
]

for filename in files_to_copy:
    source_path = os.path.join('/content', filename) # Files are in /content
    destination_path = os.path.join(target_drive_path, filename)
    try:
        shutil.copy(source_path, destination_path)
        print(f"   - Copied {filename} to Drive")
    except FileNotFoundError:
        print(f"   - Warning: {filename} not found locally, skipping copy to Drive.")
    except Exception as e:
        print(f"   - Error copying {filename} to Drive: {e}")
```

```
print("\n✅ Files saved to Google Drive!")
```

```
Saving files to: /content/drive/MyDrive/Fall 2025/Foundations of Artificial Intelli
  - Copied final_results_table.csv to Drive
  - Copied model_comparison.png to Drive
  - Copied ordinal_vs_nominal.png to Drive

✅ Files saved to Google Drive!
```

✅ Project Complete!

Output Files:

1. `class_distribution.png` - For Dataset section
2. `confusion_matrices_nominal.png` - Nominal model results
3. `confusion_matrices_ordinal.png` - Ordinal model results
4. `model_comparison.png` - Main comparison figure
5. `ordinal_vs_nominal.png` - Encoding comparison
6. `final_results_table.csv` - Results table for report

For Report:

- Use the summary text above in your Discussion section
- Include all figures in Results section
- Cite the dataset properly

Citation:

```
@article{hou2024bridging,
  title={Bridging Language and Items for Retrieval and Recommendation},
  author={Hou, Yupeng and Li, Jiacheng and He, Zhankui and Yan, An and Chen, Xiusi an
  journal={arXiv preprint arXiv:2403.03952},
  year={2024}
}
```