test_model_eval

March 16, 2026

1 Test Set Evaluation: Final Model Assessment

This notebook applies our four adopted tuned models to the test holdout set (2021-2025) that has never been used during training or tuning. This is the definitive evaluation of whether our models generalize to unseen future data, and the final answer to our research question: do weather conditions predict railroad accident frequency?

We use Mean Poisson Deviance (MPD) as the primary evaluation metric because it is the theoretically correct loss function for count data. MPD accounts for the proportional nature of count errors (predicting 10 vs 12 is a smaller relative error than 1 vs 3). MAE and RMSE are reported as secondary metrics.

The test period (2021-2025) is particularly interesting because it includes the post-COVID recovery period. If our models trained on 1975-2014 and validated on 2015-2020 still perform well on 2021-2025 data, that confirms the temporal split was effective and our findings are robust.

[1]:
```python
import json
import pickle
import sys
import warnings
from pathlib import Path

import matplotlib.pyplot as plt
import matplotlib.style as style
import numpy as np
import pandas as pd
import statsmodels.api as sm
import yaml

sys.path.insert(0, "..")
from src.eval_utils import (
    MODEL_STYLES,
    MODEL_ORDER,
    TUNED_MODEL_STYLES,
)

warnings.filterwarnings("ignore", category=FutureWarning)
style.use("tableau-colorblind10")
```

```
plt.rcParams["figure.dpi"] = 120
plt.rcParams["font.size"] = 10
```

1.1 Load Test Data and Tuned Models

We load the test holdout set and apply each tuned model using the same preprocessing that was used during tuning. No model has ever seen this data.

[2]:
```
# Load params
with open("../params.yaml") as f:
    params = yaml.safe_load(f)

target_col = params["target"]
feature_cols = (
    params["features"]["weather"]
    + params["features"]["exposure"]
    + params["features"]["temporal"]
)

# Load test set
test = pd.read_csv("../data/splits/TEST-holdout.csv.gz")
y_test = test[target_col].copy()

print(f"Test set: {len(test):,} region-months")
print(f"Period: {test['year'].min()}-{test['year'].max()}")
print(f"Regions: {test['noaa_region'].nunique()}")
print(f"Target range: {y_test.min()} to {y_test.max()} (mean: {y_test.mean():.
 ↪1f})")
```

```
Test set: 503 region-months
Period: 2021-2025
Regions: 9
Target range: 1 to 56 (mean: 18.5)
```

[3]:
```
# Load tuned model pickles and generate predictions on test set
models_path = Path("../models/trained")
results_path = Path("../results")

test_predictions = {}

# --- XGBoost (tuned, no mean encoding) ---
X_test_xgb = test[feature_cols].copy()
with open(models_path / "xgboost_tuned_model.pkl", "rb") as f:
    xgb_model = pickle.load(f)
test_predictions["xgboost"] = np.maximum(xgb_model.predict(X_test_xgb), 0)

# --- Random Forest (tuned, no mean encoding, with imputation) ---
X_test_rf = test[feature_cols].copy()
```

```python
with open(models_path / "random_forest_imputer.pkl", "rb") as f:
    rf_imputer = pickle.load(f)
X_test_rf_clean = pd.DataFrame(
    np.asarray(rf_imputer.transform(X_test_rf)),
    columns=feature_cols,
    index=X_test_rf.index,
)
with open(models_path / "random_forest_tuned_model.pkl", "rb") as f:
    rf_model = pickle.load(f)
test_predictions["random_forest"] = np.maximum(rf_model.
↪predict(X_test_rf_clean), 0)

# --- Poisson (tuned with feature selection + interactions) ---
with open(models_path / "poisson_tuned_model.pkl", "rb") as f:
    poi_artifacts = pickle.load(f)
poi_model = poi_artifacts["model"]
poi_imputer = poi_artifacts["imputer"]
poi_scaler = poi_artifacts["scaler"]
poi_selected = poi_artifacts["selected_features"]
poi_interactions = poi_artifacts.get("interaction_features", [])

X_test_poi = test[poi_selected].copy()
for int_feat in poi_interactions:
    parts = int_feat.split("_x_")
    if len(parts) == 2 and parts[0] in test.columns and parts[1] in test.
↪columns:
        X_test_poi[int_feat] = test[parts[0]] * test[parts[1]]
X_test_poi_imputed = pd.DataFrame(
    poi_imputer.transform(X_test_poi),
    columns=X_test_poi.columns,
    index=X_test_poi.index,
)
X_test_poi_scaled = pd.DataFrame(
    poi_scaler.transform(X_test_poi_imputed),
    columns=X_test_poi_imputed.columns,
    index=X_test_poi_imputed.index,
)
test_predictions["poisson"] = np.maximum(poi_model.predict(X_test_poi_scaled),␣
↪0)

# --- Negative Binomial (tuned with feature selection + interactions) ---
with open(models_path / "neg_binomial_tuned_model.pkl", "rb") as f:
    nb_artifacts = pickle.load(f)
nb_model = nb_artifacts["model"]
nb_scaler = nb_artifacts["scaler"]
nb_medians = nb_artifacts["train_medians"]
nb_features = nb_artifacts["features"]
```

```
nb_interactions = nb_artifacts.get("interaction_features", [])

# Use the scaler's feature_names_in_ as the source of truth for columns
nb_scaler_features = nb_scaler.feature_names_in_.tolist()
nb_base_cols = [c for c in nb_scaler_features if "_x_" not in c]
X_test_nb = test[nb_base_cols].copy()
for int_feat in nb_scaler_features:
    if "_x_" in int_feat:
        parts = int_feat.split("_x_")
        if len(parts) == 2 and parts[0] in test.columns and parts[1] in test.
 ↪columns:
            X_test_nb[int_feat] = test[parts[0]] * test[parts[1]]
X_test_nb = X_test_nb[nb_scaler_features].fillna(nb_medians)
X_test_nb_scaled = pd.DataFrame(
    nb_scaler.transform(X_test_nb),
    columns=X_test_nb.columns,
    index=X_test_nb.index,
)
nb_feat_available = [c for c in nb_features if c in X_test_nb_scaled.columns]
X_test_nb_final = sm.add_constant(X_test_nb_scaled[nb_feat_available])
test_predictions["negative_binomial"] = np.maximum(nb_model.
 ↪predict(X_test_nb_final), 0)

print("Test prediction ranges:")
for key in MODEL_ORDER:
    p = test_predictions[key]
    print(f"  {TUNED_MODEL_STYLES[key]['label']:25s}  "
          f"min={p.min():.1f}  max={p.max():.1f}  mean={p.mean():.1f}")
```

```
Test prediction ranges:
  XGBoost (tuned)             min=1.7  max=46.4  mean=17.0
  Random Forest (tuned)       min=2.3  max=42.3  mean=17.5
  Poisson (tuned)             min=5.8  max=35.9  mean=16.6
  Neg. Binomial (tuned)       min=3.1  max=29.0  mean=12.7
```

1.2 Validation vs Test: Did Performance Hold?

This is the most important table in the entire project. We compare each model's performance on the validation set (2015-2020, used during tuning) against the test set (2021-2025, never seen before).

What we are testing: Whether our models generalize to future, unseen data or whether they overfit to patterns specific to the validation period.

What to look for: If test MPD is close to validation MPD (within 10-15%), the model generalizes well. A large gap means the model learned something specific to 2015-2020 that doesn't transfer. The test period includes post-COVID recovery, so some degradation is expected.

[4]:

```
from sklearn.metrics import mean_absolute_error, mean_squared_error,␣
↪mean_poisson_deviance

# Load validation tuned metrics for comparison
val_metrics = {}
val_files = {
    "xgboost": "xgboost_tuned_metrics.json",
    "random_forest": "random_forest_tuned_metrics.json",
    "poisson": "poisson_tuned_interactions_metrics.json",
    "negative_binomial": "neg_binomial_tuned_interactions_metrics.json",
}
for key, fname in val_files.items():
    with open(results_path / fname) as f:
        val_metrics[key] = json.load(f)

# Compute test metrics
test_metrics = {}
for key in MODEL_ORDER:
    preds = test_predictions[key]
    preds_clipped = np.maximum(preds, 1e-6)  # MPD needs positive predictions
    test_metrics[key] = {
        "mae": mean_absolute_error(y_test, preds),
        "rmse": np.sqrt(mean_squared_error(y_test, preds)),
        "mpd": mean_poisson_deviance(y_test, preds_clipped),
    }

# Print comparison table
print("=" * 95)
print("VALIDATION vs TEST PERFORMANCE (primary metric: Mean Poisson Deviance)")
print("=" * 95)
print(f"\n{'Model':20s}  {'Val MPD':>8s}  {'Test MPD':>9s}  {'Delta':>8s}  "
      f"{'Val MAE':>8s}  {'Test MAE':>9s}  {'Val RMSE':>9s}  {'Test RMSE':
↪>10s}")
print("-" * 95)

for key in MODEL_ORDER:
    s = MODEL_STYLES[key]
    v = val_metrics[key]["validation_full"]
    t = test_metrics[key]
    mpd_delta = t["mpd"] - v["mean_poisson_deviance"]
    print(f"  {s['label']:18s}  {v['mean_poisson_deviance']:8.2f}  {t['mpd']:9.
↪2f}  "
          f"{mpd_delta:+8.2f}  {v['mae']:8.2f}  {t['mae']:9.2f}  "
          f"{v['rmse']:8.2f}  {t['rmse']:10.2f}")

print("-" * 95)
print("Positive delta = test performance is worse than validation.")
```

```
=================================================================================
===============
VALIDATION vs TEST PERFORMANCE (primary metric: Mean Poisson Deviance)
=================================================================================
===============

Model                   Val MPD   Test MPD     Delta   Val MAE   Test MAE   Val
RMSE   Test RMSE
---------------------------------------------------------------------------------
---------------
  XGBoost                  3.54       3.72     +0.18      5.98       6.10
8.62        8.73
  Random Forest            4.16       4.29     +0.13      6.42       6.63
9.21        9.28
  Poisson                  8.76       9.03     +0.27     10.30      10.66
12.76       13.19
  Neg. Binomial           10.07      10.94     +0.86     10.39      10.59
13.66       14.32
---------------------------------------------------------------------------------
---------------
Positive delta = test performance is worse than validation.
```

1.3 Test Set: Mean Poisson Deviance by Model

What we are visualizing: A bar chart of MPD on the test set for all four models. This is the single most important chart because MPD is the correct metric for count data and this is the data the models have never seen.

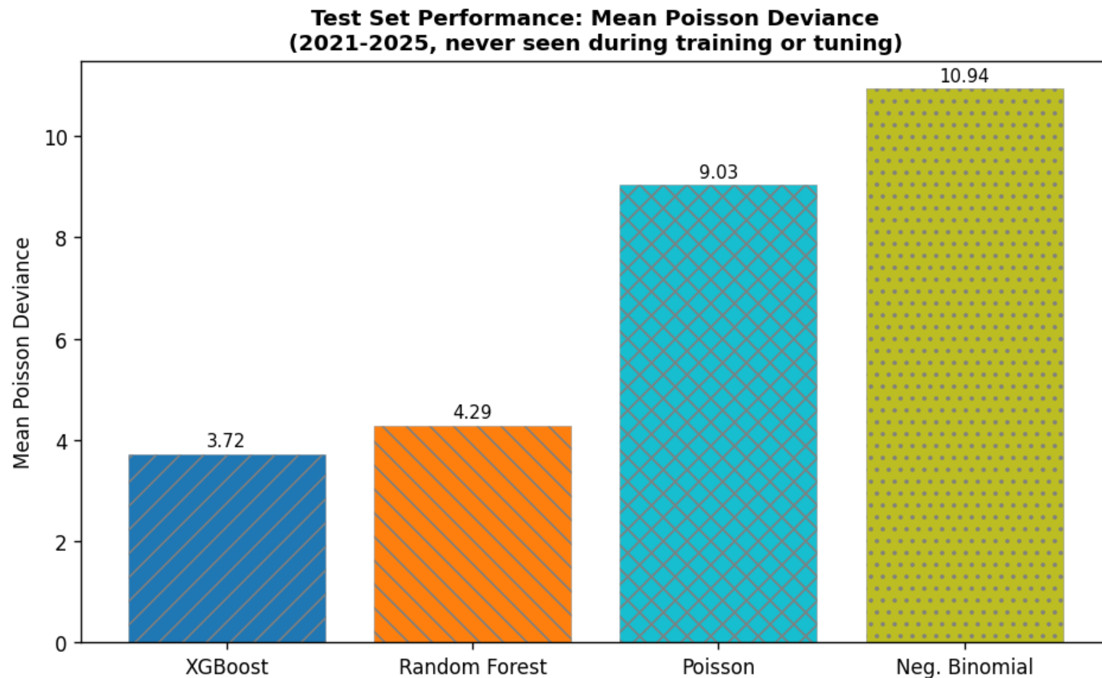
Why this visual: A simple bar chart makes the model ranking immediately clear. No paired comparison needed here since validation results were shown in the table above.

What to look for: Which model has the lowest MPD? Did the model ranking change from validation to test? If XGBoost still leads, the tree model advantage is confirmed on truly unseen data.

[5]:

```python
fig, ax = plt.subplots(figsize=(8, 5))

x = np.arange(len(MODEL_ORDER))
bars = []
for i, key in enumerate(MODEL_ORDER):
    s = MODEL_STYLES[key]
    bar = ax.bar(x[i], test_metrics[key]["mpd"], color=s["colour"],
                 edgecolor="gray", linewidth=0.3, hatch=s.get("hatch", ""))
    bars.append(bar)
    ax.text(x[i], test_metrics[key]["mpd"] + 0.15,
            f"{test_metrics[key]['mpd']:.2f}", ha="center", fontsize=9)

ax.set_xticks(x)
ax.set_xticklabels([MODEL_STYLES[k]["label"] for k in MODEL_ORDER])
```

```
ax.set_ylabel("Mean Poisson Deviance")
ax.set_title("Test Set Performance: Mean Poisson Deviance\n(2021-2025, never␣
↪seen during training or tuning)",
             fontsize=11, fontweight="bold")
plt.tight_layout()
plt.show()
```



1.4 Predicted vs Actual: Test Set

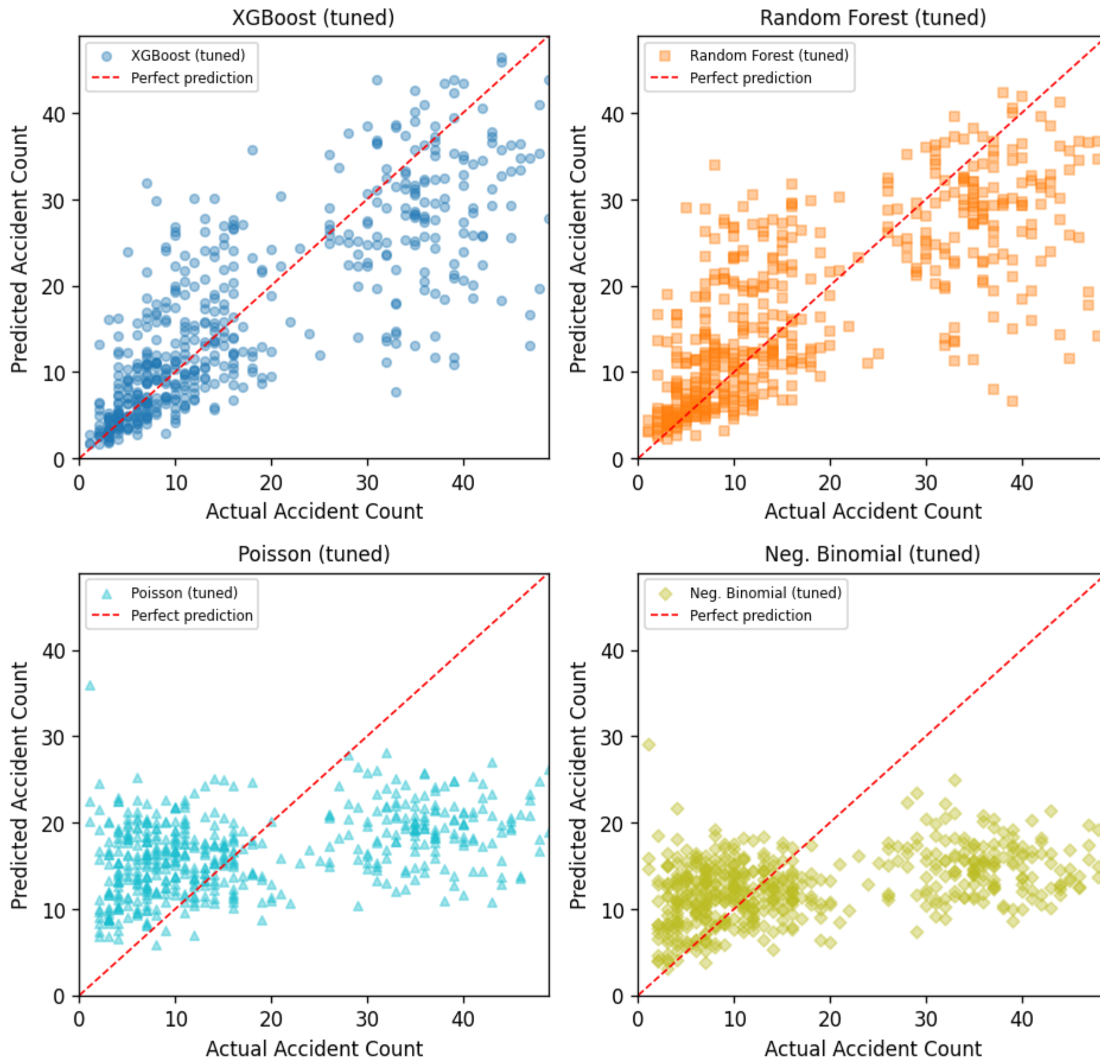
What we are testing: Whether each model's predictions track actual accident counts across the full range of values in the test set. This is the visual proof that the models generalize.

Why this visual: Scatter plots expose systematic bias (consistent over- or under-prediction), range compression (predictions clustered in the middle), and outlier behavior. All four panels share the same axis scale for fair comparison.

What to look for: Points along the red diagonal indicate accurate predictions. Points above the line mean the model overpredicts, below means underprediction. If the GLMs still show the compressed horizontal band seen during validation, that confirms the limitation is structural, not a validation artifact.

[6]:
```
fig, axes = plt.subplots(2, 2, figsize=(8, 8))
axes = axes.flatten()

all_vals = np.concatenate([np.array(y_test)] + [p for p in test_predictions.
↪values()])
```

```
axis_max = np.percentile(all_vals, 99) * 1.1

for idx, model_key in enumerate(MODEL_ORDER):
    ax = axes[idx]
    s = MODEL_STYLES[model_key]
    preds = test_predictions[model_key]

    ax.scatter(y_test, preds, alpha=0.4, s=20, color=s["colour"],
               marker=s.get("marker", "o"),␣
↪label=TUNED_MODEL_STYLES[model_key]["label"])
    ax.plot([0, axis_max], [0, axis_max], "r--", linewidth=1, label="Perfect␣
↪prediction")
    ax.set_xlim(0, axis_max)
    ax.set_ylim(0, axis_max)
    ax.set_xlabel("Actual Accident Count")
    ax.set_ylabel("Predicted Accident Count")
    ax.set_title(f"{TUNED_MODEL_STYLES[model_key]['label']}", fontsize=10)
    ax.legend(fontsize=7, loc="upper left")

fig.suptitle("Predicted vs Actual: Test Set (shared axis scale)",
             fontsize=12, fontweight="bold")
plt.tight_layout()
plt.show()
```



1.5 Regional Error Heatmap: Test Set

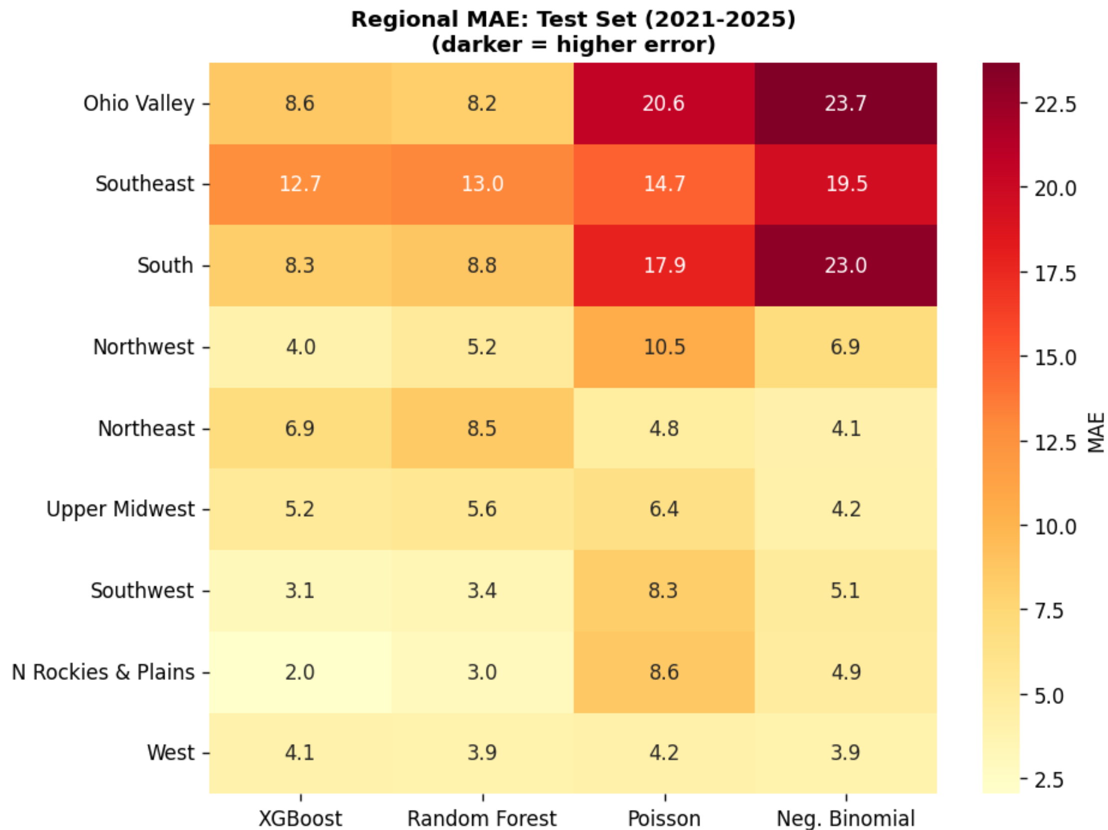
What we are testing: How well each model predicts across the 9 NOAA climate regions. Some regions have consistently higher accident counts (Ohio Valley, South, Southeast), and the question is whether the models maintain their accuracy across regions on unseen data.

Why this visual: A heatmap makes it immediately visible which model-region combinations perform well (light cells) and which struggle (dark cells). Sorting by overall difficulty puts the hardest regions at the top.

What to look for: The same high-error regions from validation (Ohio Valley, South, Southeast) should appear at the top. If a model that was accurate in a region during validation now struggles on test data, that suggests temporal instability in that region. Compare the tree model columns to the GLM columns to see if the architecture gap persists.

[7]:

```
import seaborn as sns

# Compute MAE by region for each model
region_mae = {}
for model_key in MODEL_ORDER:
    preds = test_predictions[model_key]
    for region in test["noaa_region"].unique():
        mask = test["noaa_region"] == region
        mae = mean_absolute_error(y_test[mask], preds[mask])
        if region not in region_mae:
            region_mae[region] = {}
        region_mae[region][MODEL_STYLES[model_key]["label"]] = mae

heatmap_df = pd.DataFrame(region_mae).T
heatmap_df["mean_mae"] = heatmap_df.mean(axis=1)
heatmap_df = heatmap_df.sort_values("mean_mae", ascending=False)
heatmap_df = heatmap_df.drop(columns=["mean_mae"])

fig, ax = plt.subplots(figsize=(8, 6))
sns.heatmap(heatmap_df, annot=True, fmt=".1f", cmap="YlOrRd",
            ax=ax, cbar_kws={"label": "MAE"})
ax.set_title("Regional MAE: Test Set (2021-2025)\n(darker = higher error)",
             fontsize=11, fontweight="bold")
ax.set_xlabel("")
ax.set_ylabel("")
plt.tight_layout()
plt.show()
```



1.6 Time Series: Actual vs Predicted (Top 3 Regions)

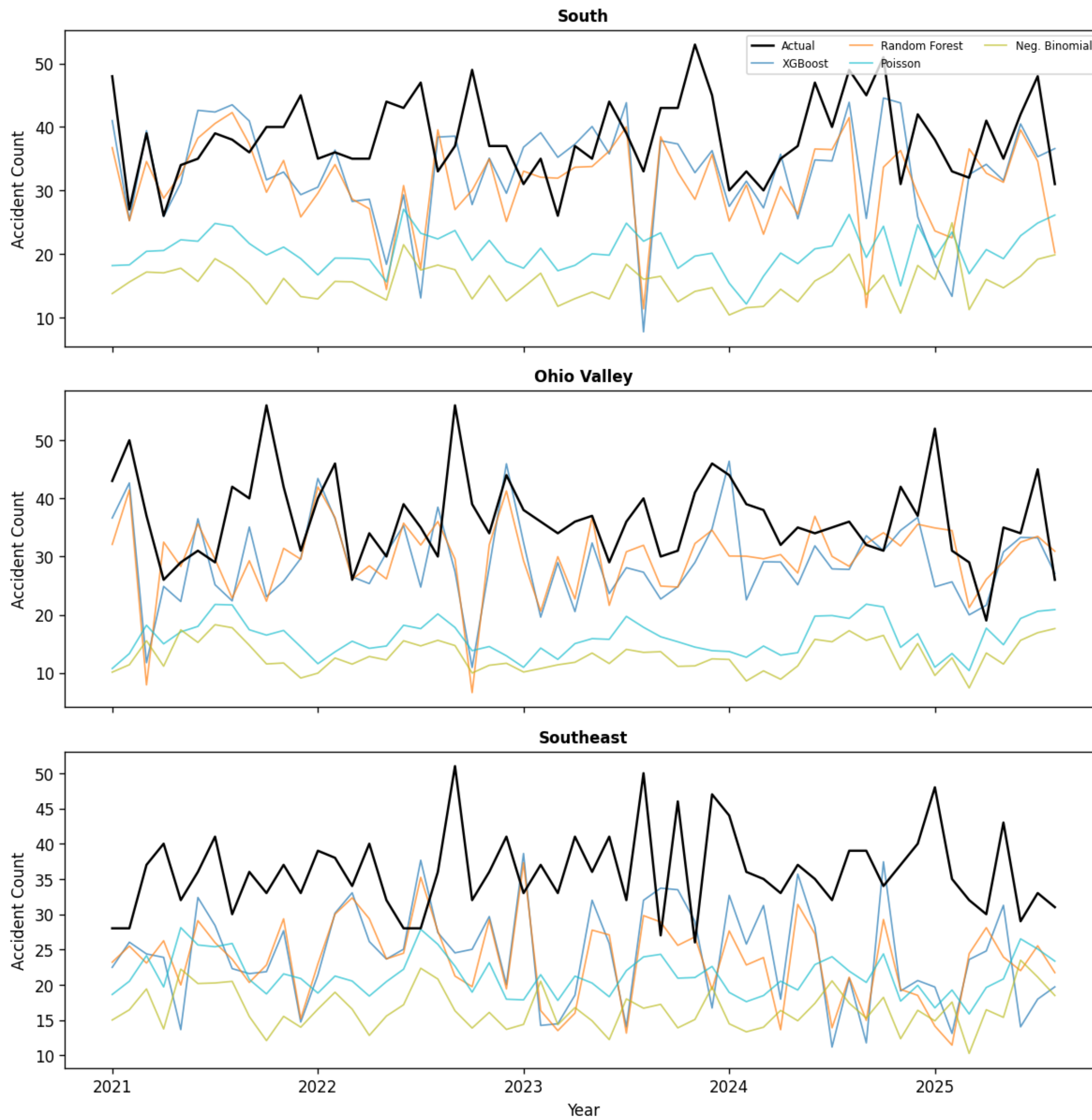
What we are testing: Whether the models track monthly accident patterns over time, not just get the average right. This reveals whether models capture seasonal variation and year-to-year trends on unseen data.

Why this visual: A time series is the only way to see temporal patterns that scatter plots and summary metrics hide. If the model gets the right average but misses seasonal peaks, that's a different failure mode than getting the average wrong.

What to look for: The predicted lines should follow the actual line's shape (peaks in winter, troughs in summer) even if they don't match perfectly. XGBoost and RF should track better than the GLMs. If a model produces a flat line, it's not capturing temporal variation at all.

[8]:
```
# Select top 3 regions by accident count for time series
top_regions = (test.groupby("noaa_region")["accident_count"]
               .mean().sort_values(ascending=False).head(3).index.tolist())

fig, axes = plt.subplots(len(top_regions), 1, figsize=(10, 3.5 *␣
 ↪len(top_regions)),
                          sharex=True)
```

```
for i, region in enumerate(top_regions):
    ax = axes[i]
    mask = test["noaa_region"] == region
    region_data = test[mask].sort_values(["year", "month"])
    time_index = range(len(region_data))

    # Plot actual
    ax.plot(time_index, region_data["accident_count"].values,
            color="black", linewidth=1.5, label="Actual", zorder=5)

    # Plot each model's predictions
    for model_key in MODEL_ORDER:
        s = MODEL_STYLES[model_key]
        # Use the boolean mask to get predictions in original order,
        # then sort to match the sorted region data
        region_indices = region_data.index
        sorted_preds = test_predictions[model_key][region_indices]
        ax.plot(time_index, sorted_preds,
                color=s["colour"], linewidth=1, alpha=0.7,
                label=s["label"])

    ax.set_title(f"{region}", fontsize=10, fontweight="bold")
    ax.set_ylabel("Accident Count")
    if i == 0:
        ax.legend(fontsize=7, loc="upper right", ncol=3)

# X-axis labels
tick_positions = []
tick_labels = []
region_data_ref = test[test["noaa_region"] == top_regions[0]].
 ↪sort_values(["year", "month"])
for j, (_, row) in enumerate(region_data_ref.iterrows()):
    if row["month"] == 1:
        tick_positions.append(j)
        tick_labels.append(str(int(row["year"])))
axes[-1].set_xticks(tick_positions)
axes[-1].set_xticklabels(tick_labels)
axes[-1].set_xlabel("Year")

fig.suptitle("Monthly Accident Count: Actual vs Predicted (Test Set, Top 3␣
 ↪Regions)",
             fontsize=12, fontweight="bold")
plt.tight_layout()
plt.show()
```



1.7 Summary

[9]:
```
print("=" * 65)
print("TEST SET EVALUATION SUMMARY")
print("=" * 65)

for metric, label in [("mpd", "Mean Poisson Deviance"), ("mae", "MAE"),␣
↪("rmse", "RMSE")]:
    best_key = min(MODEL_ORDER, key=lambda k: test_metrics[k][metric])
    print(f"\n  Lowest {label}: {MODEL_STYLES[best_key]['label']}␣
↪({test_metrics[best_key][metric]:.2f})")
```

```
print(f"\n  Test period: 2021-2025")
print(f"  Test rows: {len(y_test):,} region-months")
print(f"  Actual accident count range: {y_test.min()} to {y_test.max()}")
print("=" * 65)
```

```
=================================================================
TEST SET EVALUATION SUMMARY
=================================================================

  Lowest Mean Poisson Deviance: XGBoost (3.72)

  Lowest MAE: XGBoost (6.10)

  Lowest RMSE: XGBoost (8.73)

  Test period: 2021-2025
  Test rows: 503 region-months
  Actual accident count range: 1 to 56
=================================================================
```

1.8 Observations and Conclusions

[10]:

```
# Compute validation vs test gaps for narrative
print("Validation-to-Test MPD change:")
for key in MODEL_ORDER:
    v_mpd = val_metrics[key]["validation_full"]["mean_poisson_deviance"]
    t_mpd = test_metrics[key]["mpd"]
    pct_change = ((t_mpd - v_mpd) / v_mpd) * 100
    print(f"  {MODEL_STYLES[key]['label']:18s}  Val: {v_mpd:.2f}  Test: {t_mpd:.
 ↪2f}  Change: {pct_change:+.1f}%")
```

```
Validation-to-Test MPD change:
  XGBoost             Val: 3.54  Test: 3.72  Change: +5.1%
  Random Forest       Val: 4.16  Test: 4.29  Change: +3.1%
  Poisson             Val: 8.76  Test: 9.03  Change: +3.1%
  Neg. Binomial       Val: 10.07  Test: 10.94  Change: +8.6%
```