LTE Traffic Prediction with Machine Learning: A Comparative Study

It's an excellent idea to leverage the "Beyond Throughput: a 4G LTE Dataset with Channel and Context Metrics" dataset for your Streamlit project, combining it with Machine Learning and Deep Learning. The sources you've provided offer a rich foundation for various types of analyses.

Here's an orientation for developing your project, structured for a multi-page Streamlit application, drawing on the insights from the provided papers:

First, let's understand the "Beyond Throughput" dataset:
This dataset is composed of client-side cellular Key Performance Indicators (KPIs) collected from two major Irish mobile operators across different mobility patterns: static, pedestrian, car, bus, and train. It includes 135 traces with an average duration of fifteen minutes per trace, offering throughput ranging from 0 to 173 Mbit/s at one-second granularity.
Key metrics captured include:

- **Throughput**: `DL_bitrate` (download rate) and `UL_bitrate` (uplink rate) measured at the application layer.
- **Channel-related KPIs**: RSRQ, RSRP, RSSI, SNR, CQI, and values for neighboring cells (NRxRSRQ & NRxRSRP). These describe the radio environment.
- **Context-related metrics**: Timestamp, Longitude and Latitude (GPS coordinates of mobile device and serving eNodeB), Velocity, Operator name, CellId, NetworkMode, and Distance between serving cell and mobile device.
- It also contains information about handover events.
  The dataset also acknowledges limitations such as one-second sampling granularity and potential missing data for some records (e.g., RSSI, eNodeB geo-locations).

Given this, here are several analysis types you could implement as separate pages in your Streamlit application:

Streamlit App Structure and Advised Analyses:

Page 1: Data Overview & Exploration

This page would introduce the dataset and allow initial exploration.

- **Purpose**: Provide users with a foundational understanding of the data.
- **Content**:
    - Brief description of the "Beyond Throughput" dataset and its origin.
    - Display a sample of the raw data [Table 1 in 340].
    - Summary statistics (mean, median, standard deviation) for key numerical features (e.g., `DL_bitrate`, `RSRP`, `SINR`, `Velocity`) [Table 2, 3 in 60, 65].
    - Visualizations:
        - Distribution of `DL_bitrate` and `UL_bitrate` across different mobility patterns and operators [Figure 1 in 59, Table 2 in 60].
        - Geographic visualization of the routes (using `Longitude` and `Latitude`) to show data collection paths, potentially distinguishing by mobility pattern or operator [Figure 3 in 65].
        - Correlation heatmap of all relevant features to understand relationships between network parameters and throughput [Figure 3 in 22, Figure 6 in 143, Table 2 in 353]. This helps identify multicollinearity.
- **User Controls**: Dropdowns to select mobility pattern, operator, or specific features for visualization.

Page 2: Throughput Prediction (Regression Task)

This is a central focus in several provided papers.

- **Purpose**: Predict Downlink (DL) or Uplink (UL) throughput using other network and context metrics.
- **Methodology**:
    - **Data Preprocessing**:
        - Handle missing values (e.g., using imputation methods).
        - Categorical features (`CellId`, `Operatorname`, `NetworkMode`) should be converted to numerical format using One-Hot Encoding (OHE). Note

that OHE might be impractical for `CellId` if there are too many unique values.

        - Numerical feature scaling: Min-Max Scaling or standardization is often necessary, especially for linear models like SVM.
        - Feature engineering: Create new features from existing ones (e.g., derived features from `Timestamp` like `Hour`, `Month`, `Day`, `Year`, as done in). Apply transformations like `ln(Trafficj)` for normal distribution or sinusoidal encoding for cyclical features like `Hour`.
        - Data binning can reduce noise and smooth data, especially in drive tests.
    - **ML Models**:
        - Start with classical ML models: Support Vector Machines (SVM/SVR), Bagging, Random Forest.
        - Also include Linear Regression (LR), K-Nearest Neighbors (KNN/KNNR), Decision Tree Regression (DTR).
        - Consider more advanced tree-based models: XGBoost and CatBoost, which are known for good performance in tabular data.
        - For deep learning, a Multilayer Perceptron (MLP) or Artificial Neural Network (ANN) can be applied for tabular data.
    - **Training & Evaluation**:
        - Split data into training and testing sets (e.g., 80% train, 20% test).
        - Evaluate models using metrics like Root Mean Square Error (RMSE), Mean Absolute Error (MAE), and Coefficient of Determination (R2).
        - Hyperparameter tuning (e.g., using Grid Search with cross-validation) for each model.
- **User Controls**:
    - Dropdowns to select the target variable (`DL_bitrate` or `UL_bitrate`).
    - Selection of ML model(s) to train and compare.
    - Sliders/inputs for specific model hyperparameters (if desired, for advanced users).
    - Options to filter data by `Operatorname`, `Mobility Pattern`.

- **Outputs**:
  - Table summarizing performance metrics (R2, RMSE, MAE) for all selected models [Table 1 in 25, Table 5 in 155, Table 1 in 310, Table 3 in 353].
  - Plot of actual vs. predicted throughput for a sample trace or time period [Figure 7 in 160, Figure 3 in 312].
  - Feature importance analysis (especially for tree-based models like Random Forest, XGBoost) to show which factors most influence throughput [Table 5 in 155, Figure 7 in 239, Figure 6 in 354]. `SINR`, `p_a`, `CellLoad`, `RSRP`, `RSRQ`, and `RSSI` are consistently highlighted as important.

Page 3: Channel & Contextual Impact Analysis
This page focuses on understanding the direct relationships between various KPIs.

- **Purpose**: Visualize and quantify the impact of channel conditions and contextual information on throughput.
- **Methodology**:
  - Analyze relationships between `DL_bitrate`/`UL_bitrate` and channel KPIs (`RSRP`, `RSRQ`, `SINR`, `RSSI`, `CQI`).
  - Investigate the effect of mobility (`Velocity`) and location (`Distance` to cell, `GPS coordinates`) on throughput.
- **User Controls**:
  - Dropdowns to select pairs of features for scatter plots or correlation analysis.
  - Filters for `Mobility Pattern` and `Operatorname`.
- **Outputs**:
  - Scatter plots of `DL_bitrate` vs. `CQI`, `RSRP`, `SINR`, `RSSI` [Figure 2 in 64, Figure 2,3,4,5 in 346, 347].
  - Box plots showing throughput distribution across different `CQI` levels [Figure 2 in 64].
  - Calculated correlation coefficients (e.g., Pearson, Spearman) between selected features and throughput.

- Analysis of how average throughput and its variation range differ across operators and mobility patterns [Table 2 in 60].

Page 4: Mobility & Handover Insights
This page could explore the mobility patterns and their implications, including handover.

- **Purpose**: Analyze how user mobility influences network performance and predict mobility-related events.
- **Methodology**:
  - Analyze `Velocity` data across different mobility patterns [Table 3 in 65].
  - Utilize `CellId` and `NRxRSRP` / `NRxRSRQ` to infer handover events or potential handovers. The dataset explicitly mentions `handover events`.
  - This page can discuss the relevance of predicting handover events for Quality of Experience (QoE).
- **ML Models (for advanced sub-tasks)**:
  - Classification models to predict handover *occurrence* (e.g., a binary classification for "handover" vs. "no handover") based on channel metrics from serving and neighboring cells, and velocity.
  - The paper "c-19-7.pdf" discusses predicting handover success rates as QoE KPIs, though it notes limitations if mobility-related information like individual speed isn't available. Your "Beyond Throughput" dataset has `Velocity`, which helps.
- **User Controls**:
  - Selection of mobility pattern for analysis.
  - Visualization of velocity trends over time for selected traces.
- **Outputs**:
  - Plots showing changes in RSRP/RSRQ from serving and neighboring cells over time for traces with handovers.
  - Metrics for handover prediction model performance (if implemented).

Page 5: Time Series Forecasting (Advanced)
This page would delve into predicting future network performance.

- **Purpose**: Predict future throughput values or other KPIs based on historical time-series data.
- **Methodology**:
  - Focus on `DL_bitrate` or `UL_bitrate` as time series.
  - **Models**:
    - PROPHET (Facebook's forecasting model): Good for capturing hourly, daily, and weekly seasonality, and trends.
    - Recurrent Neural Networks (RNN), specifically Long Short-Term Memory (LSTM) networks, are well-suited for time-series data.
    - ARIMA models are also an option for time series forecasting.
  - **Training**: Train models on a rolling window of historical data or a fixed period.
  - **Evaluation**: Compare predicted values against actual future values using RMSE.
- **User Controls**:
  - Select a specific cell or operator for time series analysis.
  - Choose a prediction horizon (e.g., next 1 hour, next 24 hours, next week).
  - Select the time series model (e.g., PROPHET, LSTM).
- **Outputs**:
  - Plots showing actual vs. predicted throughput over time, including uncertainty intervals (if applicable) [Figure 8 in 239].
  - RMSE values for different prediction horizons and models [Table 1 in 240].
  - Visualizations of trend and seasonality components extracted by PROPHET.

Possible Future Expansion (Beyond Initial Project Scope):

- **Synthetic Data Generation (GANs)**: If you find the dataset size limiting for certain deep learning models, you could explore Generative Adversarial Networks (GANs) to create synthetic data, as discussed in. This would involve training a GAN on your existing dataset to generate more realistic samples, which could then augment your training data. This is a complex task and might warrant its own separate project.

General Development Advice for Streamlit:

1. **Start Small**: Begin with one page and one analysis type (e.g., simple throughput prediction with LR or KNN) to get the Streamlit flow working.
2. **Data Loading**: Load the dataset efficiently. For large datasets, consider using `st.cache_data` to avoid re-loading on every interaction.
3. **Preprocessing Pipeline**: Implement your data preprocessing steps in a modular and reusable way. You might have a dedicated Python module for data cleaning and feature engineering that your Streamlit app imports.
4. **Model Saving/Loading**: Once models are trained, save them (e.g., using `joblib` or `pickle`) and load them in your Streamlit app. For ML models, you don't want to retrain them every time the app runs.
5. **Interactivity**: Use Streamlit widgets (sliders, checkboxes, selectboxes, text inputs) to allow users to interact with your models and visualizations.
6. **Clear Explanations**: For each page, clearly explain the purpose, the models used, the metrics displayed, and the interpretation of the results. This is crucial for user understanding, especially for a technical project.
7. **Error Handling**: Consider what happens if a user makes an unexpected selection or if data is missing for a particular calculation.
8. **Performance**: For computationally intensive tasks (like training large ML models or extensive data processing), consider:
   - Pre-training models and loading them.
   - Using `st.spinner` to show loading states.
   - Optimizing your Python code for speed.

By following this structured approach, you can build a comprehensive and insightful Streamlit application that effectively utilizes the "Beyond Throughput" dataset and the machine learning techniques discussed in your provided sources. Good luck with your project!