05_Modeling_Evaluation

June 8, 2025

1 Predicting ICU Length of Stay

In this chapter, we transition from data preprocessing to predictive modeling, with the objective of accurately estimating the Length of Stay (LOS) in the Intensive Care Unit (ICU). The modeling phase is a critical step that leverages the engineered features and cleaned dataset constructed in the previous stages. Our approach is structured in increasing complexity, starting from simple interpretable models to more flexible and high-performance machine learning techniques.

We begin by splitting the dataset into training and testing subsets to ensure proper evaluation of generalization. Baseline models such as Linear Regression and Decision Trees are first employed to establish reference performance metrics. Subsequently, we extend the analysis to ensemble methods like Random Forests and gradient-boosting algorithms (e.g., XGBoost), which are particularly suitable for handling nonlinear relationships and mixed data types.

Performance is rigorously assessed using multiple regression metrics, including Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R-squared (R^2). These metrics provide complementary insights into model accuracy, robustness, and explanatory power. Finally, we explore hyperparameter optimization and cross-validation strategies to enhance model reliability and generalizability.

This modeling chapter thus represents the computational core of the study and is essential for translating raw ICU data into clinically actionable predictions.

[ ]:
```python
# === Essential Libraries ===
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os
from sklearn.model_selection import train_test_split
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

```
[ ]: # === Configuration Constants ===
     EXPORT_PATH = "../content/"
     # === Load dataset ===
     df_final = pd.read_csv(EXPORT_PATH + "df_final_processed.csv")

     # === Confirm structure ===
     print(df_final.shape)
     display(df_final.head())
     df_final.isnull().sum().sort_values(ascending=False)/len(df_final)
```

```
(3685, 71)

        AGE  GENDER     LOS  HOSPITAL_EXPIRE_FLAG  HEART_RATE_MEAN  \
0  0.439560       1  3.2788                     0         0.715383
1  0.901099       1  7.1314                     1         0.151088
2  0.626374       0  0.8854                     1         0.368821
3  0.835165       0  2.4370                     1         0.468597
4  0.626374       1  3.0252                     0         0.289943

   HEART_RATE_STD  HEART_RATE_MIN  HEART_RATE_MAX  HEART_RATE_COUNT  \
0        0.080933        0.733333        0.102825          0.021082
1        0.054718        0.370370        0.038418          0.015460
2        0.135338        0.000000        0.061017          0.017569
3        0.059596        0.600000        0.064407          0.018271
4        0.027484        0.511111        0.039548          0.016163

   HEART_RATE_SKEW  …  ADMISSION_LOCATION_TRANSFER FROM OTHER HEALT  \
0         0.524927  …                                         False
1         0.721928  …                                         False
2         0.284040  …                                         False
3         0.544127  …                                         False
4         0.624800  …                                         False

   ADMISSION_LOCATION_TRANSFER FROM SKILLED NUR  INSURANCE_Medicaid  \
0                                        False                True
1                                        False               False
2                                        False               False
3                                        False               False
4                                        False               False

   INSURANCE_Medicare  INSURANCE_Private  FIRST_CAREUNIT_CSRU  \
0               False              False                False
1                True              False                False
2               False               True                False
3                True              False                False
4                True              False                False

   FIRST_CAREUNIT_MICU  FIRST_CAREUNIT_NICU  FIRST_CAREUNIT_SICU  \
0                  True                 False                 False
1                 False                 False                 False
2                  True                 False                 False
3                 False                 False                  True
4                  True                 False                 False

   FIRST_CAREUNIT_TSICU
0                 False
1                 False
2                 False
3                 False
4                 False

[5 rows x 71 columns]
```

```
[ ]: SPO2_SKEW                0.486296
     SPO2_STD                 0.485753
     RESPIRATORY_RATE_SKEW    0.485210
     HEART_RATE_SKEW          0.484668
     SPO2_COUNT               0.484668
                                ...
     FIRST_CAREUNIT_CSRU      0.000000
     FIRST_CAREUNIT_MICU      0.000000
     FIRST_CAREUNIT_NICU      0.000000
     FIRST_CAREUNIT_SICU      0.000000
     FIRST_CAREUNIT_TSICU     0.000000
     Length: 71, dtype: float64
```

1.0.1 Dataset Preparation and Target Definition

In this initial step of our modeling pipeline, we define the predictors (features) and the response variable (target) for the task of predicting ICU Length of Stay (LOS). Drawing from the fully preprocessed dataset (`df_final`), we isolate the target variable `LOS`, which quantifies the duration of a patient's ICU stay in days. To ensure that no data leakage occurs, we explicitly exclude all identifying columns and those chronologically or causally related to the outcome. Specifically, this includes patient identifiers (`SUBJECT_ID`, `HADM_ID`, `ICUSTAY_ID`), direct timestamps (`INTIME`, `OUTTIME`, `ADMITTIME`, etc.), and administrative or outcome-related fields such as `HOSPITAL_EXPIRE_FLAG` and `DEATHTIME`.

After removing these columns, we inspect and address any remaining missing values in the feature matrix by imputing them with column-wise means—a pragmatic strategy in the absence of strong domain-specific imputations. This ensures that all observations are retained for model training without introducing bias from listwise deletion.

Finally, we confirm the shape of the resulting dataset. The feature matrix `X` contains 3,685 ICU admissions and 69 engineered features, while the target vector `y` contains a matching number of observations. This alignment is crucial for subsequent modeling steps, ensuring consistency in the dimensions of the input and output data.

```
[ ]: cols_to_remove = ['SUBJECT_ID', 'HADM_ID', 'ICUSTAY_ID', 'LOS',␣
      ↪'HOSPITAL_EXPIRE_FLAG', 'INTIME', 'OUTTIME', 'ADMITTIME', 'DISCHTIME',␣
      ↪'DEATHTIME', 'DOD']

     y = df_final['LOS']
     X = df_final.drop(columns=cols_to_remove, errors='ignore')

     X = X.fillna(X.mean())  # Fill NaN values with column means
     y = y.loc[X.index]

     print(f"Shape of X: {X.shape}")
     print(f"Shape of y: {y.shape}")
```

```
Shape of X: (3685, 69)
Shape of y: (3685,)
```

1.0.2 Data Partitioning Strategy

To ensure robust model development and fair evaluation, the dataset is partitioned into three disjoint subsets: training, validation, and test. This tripartite split enables not only the estimation of model parameters and tuning of hyperparameters but also the assessment of generalization on completely unseen data.

The initial split isolates 70% of the data for training, reserving the remaining 30% for further partitioning. The residual subset is then equally divided into validation and test sets, each comprising 15% of the original dataset. This strategy results in the following allocation:

- **Training set**: 2,579 ICU admissions
- **Validation set**: 553 ICU admissions
- **Test set**: 553 ICU admissions

Such a configuration strikes a balance between maximizing training data—critical for the effective fitting of deep neural networks—and retaining enough validation and test samples to support meaningful hyperparameter optimization and unbiased model evaluation, respectively. Importantly, the random seed is fixed to ensure reproducibility of the split.

```
[ ]: X_train, X_temp, y_train, y_temp = train_test_split(
         X, y, test_size=0.30, random_state=42
     )

     X_val, X_test, y_val, y_test = train_test_split(
         X_temp, y_temp, test_size=0.50, random_state=42
     )

     print(f"Train set: {X_train.shape}, {y_train.shape}")
     print(f"Validation set: {X_val.shape}, {y_val.shape}")
     print(f"Test set: {X_test.shape}, {y_test.shape}")
```

```
Train set: (2579, 69), (2579,)
Validation set: (553, 69), (553,)
```

```
Test set: (553, 69), (553,)
```

1.1 Traditional ML Models

1.1.1 Baseline: Linear Regression

As a foundational benchmark, a linear regression model is trained on the ICU dataset to establish a minimal performance reference point. The choice of linear regression is deliberate: its simplicity, interpretability, and speed make it a valuable starting point for gauging whether a more sophisticated model architecture is justified.

In this implementation, the model is fitted using the training set and then evaluated on the test set. No regularization, polynomial terms, or interaction features are included—this ensures the model serves purely as a linear approximation of the relationship between the features and ICU Length of Stay (LOS).

The resulting performance metrics on the test set are as follows:

- **Mean Absolute Error (MAE)**: 4.59 days
- **Root Mean Squared Error (RMSE)**: 67.59 days
- **R² Score**: 0.05

These values indicate that the model, while capturing some weak linear trends, is largely unable to explain the variance in ICU LOS. The exceedingly high RMSE relative to MAE suggests the presence of substantial outliers or skewness in the data distribution, which a linear model is poorly equipped to handle. The low R² score (0.05) further confirms the model's limited explanatory power. This reinforces the need for more expressive, non-linear models—such as neural networks—to adequately model this complex clinical prediction task.

```
[ ]: # === Model ===
     lr = LinearRegression()
     lr.fit(X_train, y_train)
     y_pred_lr = lr.predict(X_test)
```

```
[ ]: # === Evaluation Metrics ===
     mae_lr = mean_absolute_error(y_test, y_pred_lr)
     rmse_lr = mean_squared_error(y_test, y_pred_lr)
     r2_lr = r2_score(y_test, y_pred_lr)
     print(f"[Linear Regression] MAE: {mae_lr:.2f}, RMSE: {rmse_lr:.2f}, R²: {r2_lr:.
      ↪2f}")
```

```
[Linear Regression] MAE: 4.59, RMSE: 67.59, R²: 0.05
```

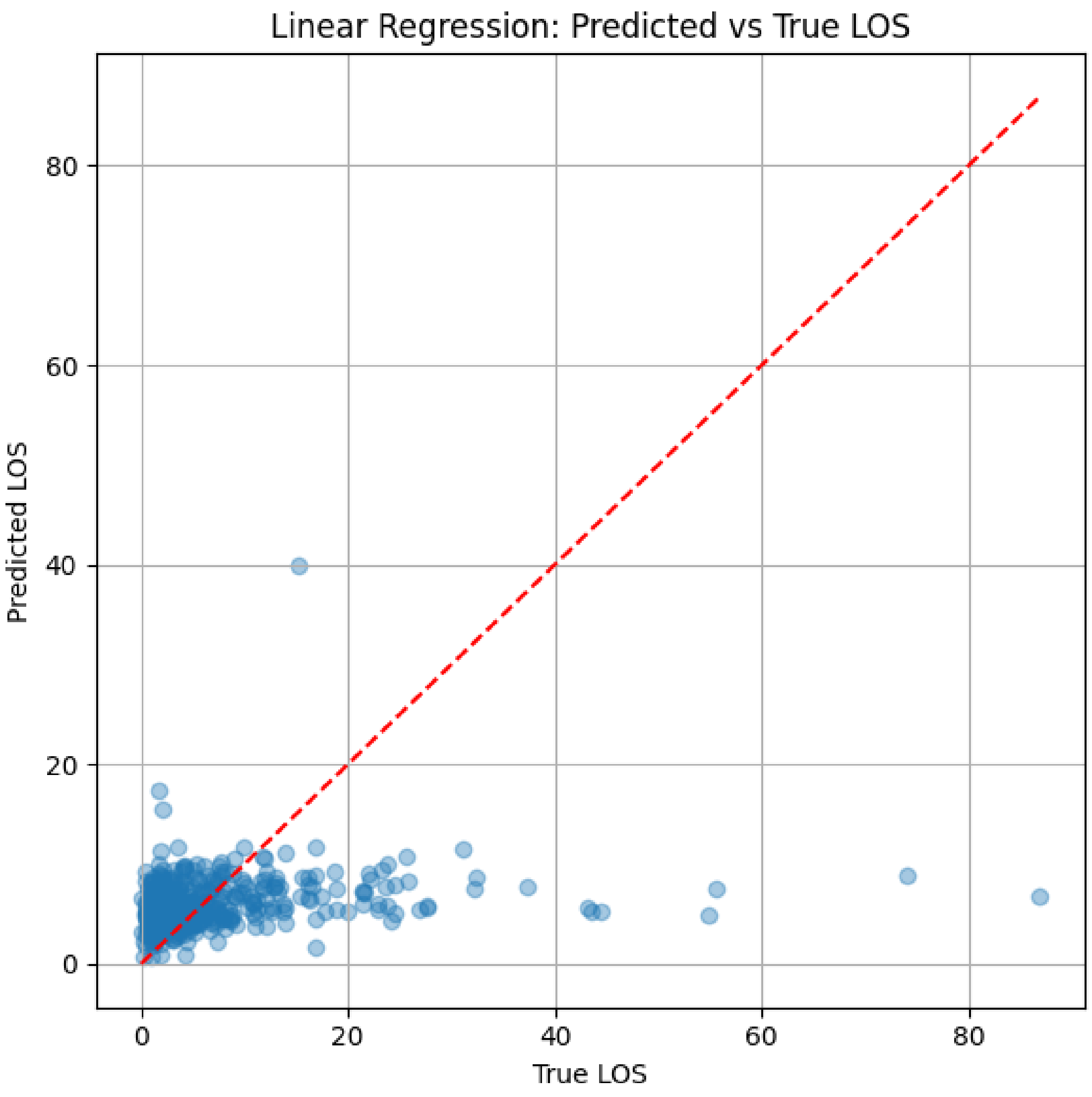
Scatter Plot Analysis for Linear Regression The scatter plot visualizes the relationship between the true ICU Length of Stay (LOS) and the values predicted by the linear regression model. Ideally, a well-calibrated model should produce points that lie close to the identity line (red dashed line), where predicted values match true observations.

In this case, the plot reveals a clear deficiency in the linear model's capacity to capture the variance of the target variable. A significant concentration of points is observed in the lower-left corner, with most predictions clustered between 0 and 20 days, regardless of the actual LOS. This suggests underestimation of long-stay patients and over-smoothing of predictions, a classic artifact of using linear models on skewed or heteroscedastic medical data.

Notably, as true LOS increases, predicted values tend to plateau, indicating that the model fails to scale its predictions in proportion to the actual outcome. This is visually evident from the divergence from the red identity line as one moves to the right side of the plot. The sparsity of points in higher LOS ranges also reflects the dataset's skewed distribution, which amplifies the difficulty for a linear estimator.

This plot provides a compelling rationale for exploring more flexible modeling techniques, such as tree-based models or deep learning architectures, which can better accommodate non-linear interactions and heterogeneity in patient trajectories.

[ ]:
```
# === Scatter Plot: True vs Predicted ===
plt.figure(figsize=(6, 6))
plt.scatter(y_test, y_pred_lr, alpha=0.4)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')
plt.xlabel("True LOS")
plt.ylabel("Predicted LOS")
plt.title("Linear Regression: Predicted vs True LOS")
plt.grid(True)
plt.tight_layout()
plt.show()
```



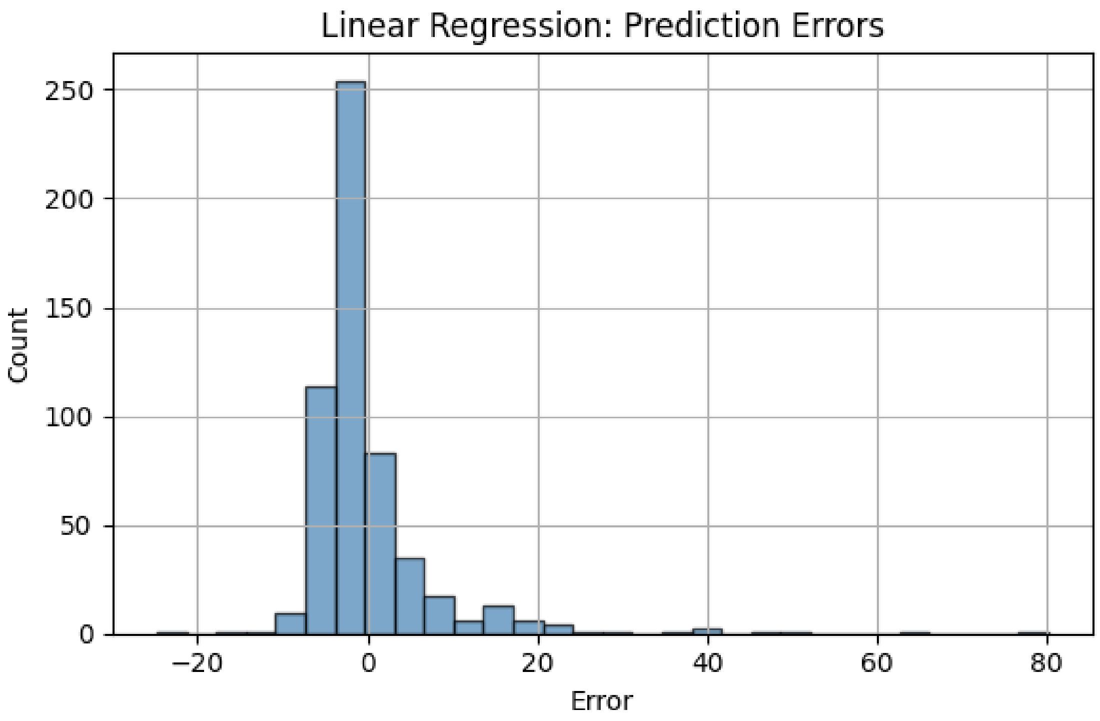
Distribution of Prediction Errors for Linear Regression The histogram above displays the distribution of prediction errors, defined as the difference between the actual and predicted LOS values. A model with unbiased predictions should ideally produce a distribution centered around zero, with errors symmetrically spread and minimal presence of extreme outliers.

In our linear regression model, the error distribution appears skewed to the right, indicating that the model tends to underpredict the length of stay, particularly in cases where the actual LOS is high. This observation is aligned with the previous scatter plot analysis, where the model failed to capture longer ICU stays.

Most errors fall within the [-5, +15] range, which suggests some degree of acceptable variance for shorter LOS, but the presence of long right-tail errors (extending beyond +40 days) is concerning. These extreme residuals reflect the model's inability to handle patients with protracted stays, likely due to its linear constraints and lack of interaction terms.

Additionally, the moderate peak near zero indicates that while some predictions are accurate, the high variance and skewness render the model unreliable for robust clinical deployment.

```
[ ]: # === Histogram of Errors ===
errors_lr = y_test - y_pred_lr
plt.figure(figsize=(6, 4))
plt.hist(errors_lr, bins=30, alpha=0.7, color="steelblue", edgecolor="k")
plt.title("Linear Regression: Prediction Errors")
plt.xlabel("Error")
plt.ylabel("Count")
plt.grid(True)
plt.tight_layout()
plt.show()
```



1.1.2 Decision Tree Regressor

Application of a basic decision tree regressor to predict ICU Length of Stay (LOS) resulted in significantly underwhelming performance metrics:

- **Mean Absolute Error (MAE): 6.79**
- **Root Mean Squared Error (RMSE): 136.11**
- **R^2 Score: -0.91**

The **negative R^2** value is particularly critical—it indicates that the model performs *worse than a simple mean predictor*, which is a strong signal of overfitting to the training data or extreme variance in predictions. This is common in unpruned decision trees, especially when applied to noisy or high-dimensional regression tasks like LOS estimation.

Additionally, the **very high RMSE** (more than double that of the linear model) suggests that the model makes frequent and severe mispredictions. Decision trees, when left unregularized, tend to create overly complex models that memorize idiosyncrasies in the training set, failing to generalize to unseen data.

In this context, the decision tree regressor demonstrates a clear inability to model ICU LOS effectively, emphasizing the necessity for either **tree pruning**, **depth constraints**, or a shift toward **ensemble methods** such as Random Forests or Gradient Boosting.

[ ]:
```
dt = DecisionTreeRegressor(random_state=42)
dt.fit(X_train, y_train)
y_pred_dt = dt.predict(X_test)
```

[ ]:
```
# === Evaluation Metrics ===
mae_dt = mean_absolute_error(y_test, y_pred_dt)
rmse_dt = mean_squared_error(y_test, y_pred_dt)
r2_dt = r2_score(y_test, y_pred_dt)
print(f"[Decision Tree] MAE: {mae_dt:.2f}, RMSE: {rmse_dt:.2f}, R²: {r2_dt:.
 ↪2f}")
```

```
[Decision Tree] MAE: 6.79, RMSE: 136.11, R²: -0.91
```

Scatter Plot Analysis for Decision Tree The scatter plot comparing **true** versus **predicted LOS** for the decision tree model highlights the model's instability and poor generalization capacity. The red dashed line represents the ideal scenario in which predicted values would perfectly match actual values (i.e., a 45-degree diagonal). However, the model's predictions appear to **cluster along discrete steps**, a common trait of decision trees due to their piecewise constant nature.

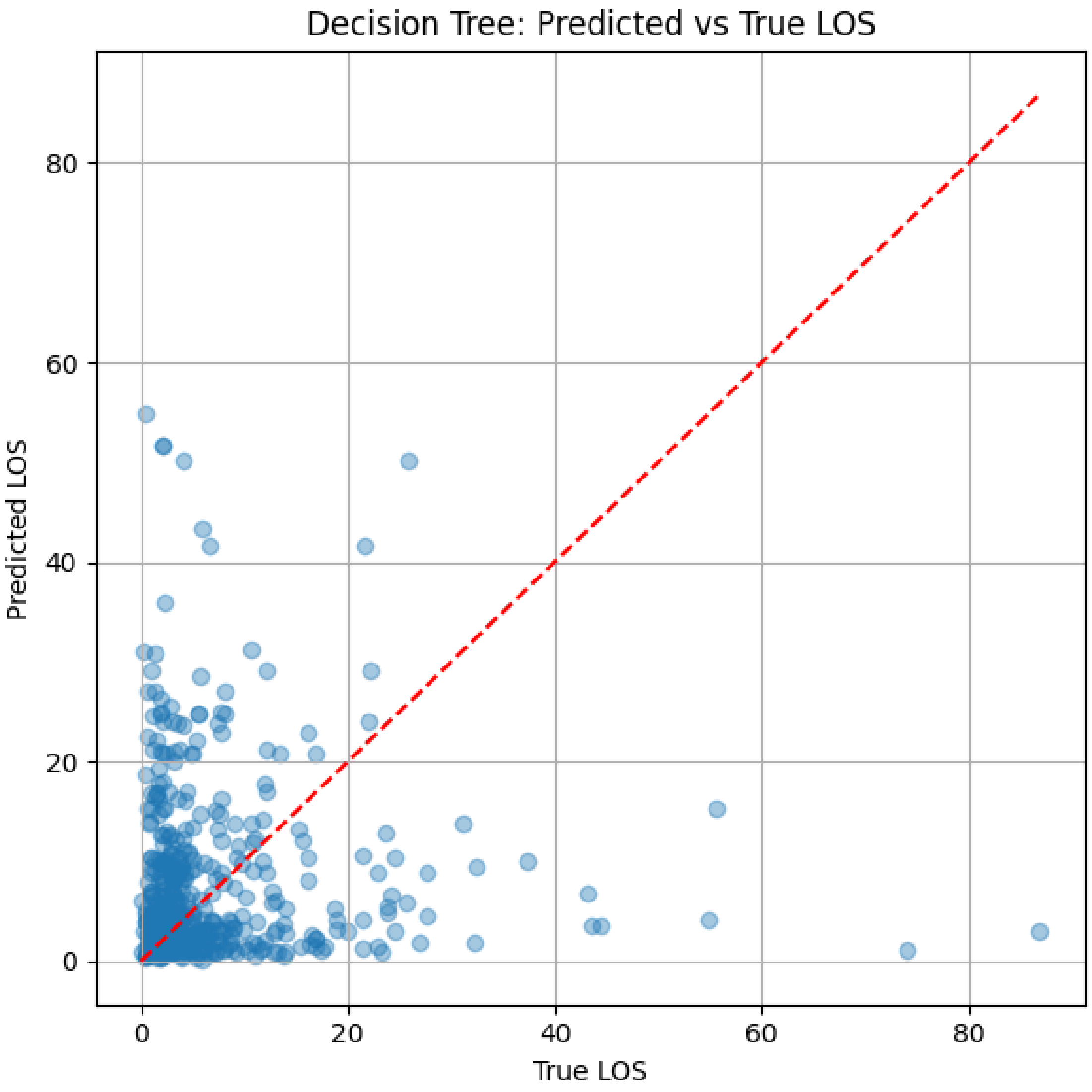
Several problematic patterns emerge:

- A **dense cluster of predictions around low LOS values**, suggesting a strong bias toward underestimating longer ICU stays.
- **Severe underestimation** of many higher LOS instances (visible as vertical stripes below the diagonal), indicating that the model fails to extrapolate for complex, long-duration cases.
- The spread of points is **asymmetrical and heteroscedastic**, with increasing variability at higher LOS values.

In essence, the plot confirms quantitatively observed issues: the model behaves adequately only for a narrow range of short-stay patients, with a **lack of predictive nuance** elsewhere. This reinforces the conclusion that unpruned decision trees are inadequate for capturing the clinical complexity of ICU LOS.

[ ]:
```
# === Scatter Plot ===
plt.figure(figsize=(6, 6))
plt.scatter(y_test, y_pred_dt, alpha=0.4)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')
plt.xlabel("True LOS")
```

```python
plt.ylabel("Predicted LOS")
plt.title("Decision Tree: Predicted vs True LOS")
plt.grid(True)
plt.tight_layout()
plt.show()
```



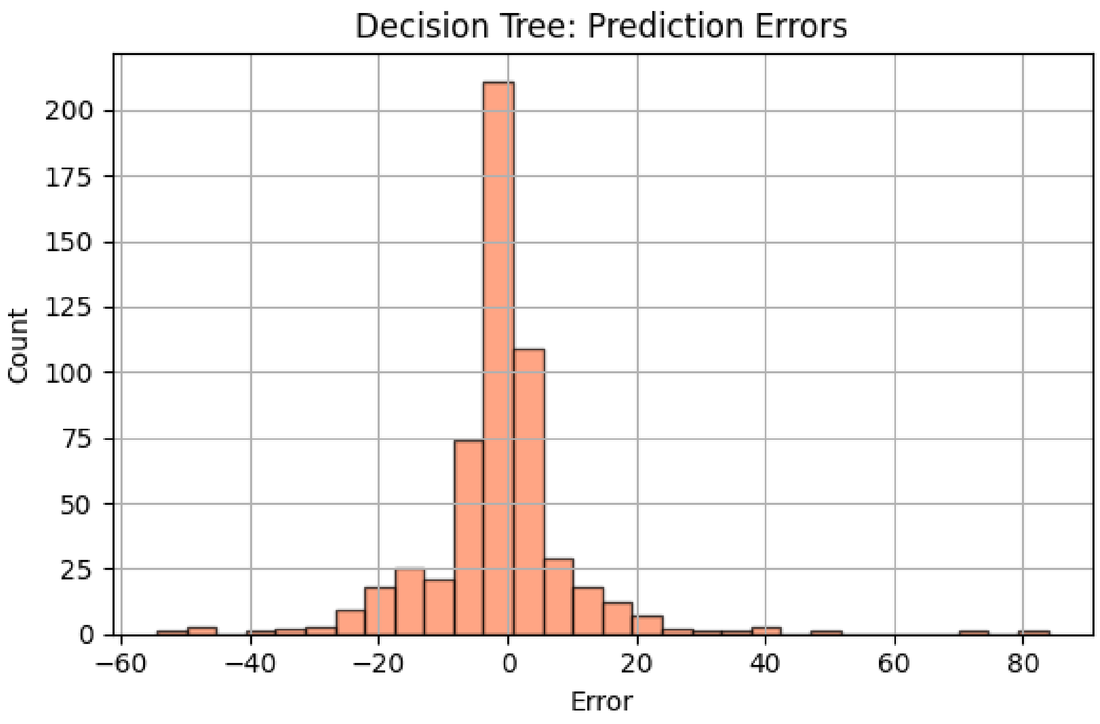
Distribution of Prediction Errors for Decision Tree The histogram of prediction errors for the decision tree model exposes a **highly dispersed and asymmetric** residual distribution. While a central peak near zero indicates that some predictions approximate the true values, the surrounding distribution displays long tails, particularly skewed towards **negative errors**—i.e., cases where the model **underestimates** the true Length of Stay (LOS).

Notable characteristics include:

- **Excess kurtosis**: The histogram is sharply peaked with fat tails, a sign of instability and overfitting to training data.
- **Bimodal tendencies** or outlier bars far from zero further support the claim that the model lacks generalization.
- The **broad dispersion of errors** indicates that the model's performance varies greatly depending on the patient profile.

This error pattern confirms that while the decision tree can capture simple patterns, it fails to model the complex, nonlinear relationships intrinsic to ICU LOS data. The results advocate for more robust and regularized models.

```
[ ]: # === Histogram of Errors ===
errors_dt = y_test - y_pred_dt
plt.figure(figsize=(6, 4))
plt.hist(errors_dt, bins=30, alpha=0.7, color="coral", edgecolor="k")
plt.title("Decision Tree: Prediction Errors")
plt.xlabel("Error")
plt.ylabel("Count")
plt.grid(True)
plt.tight_layout()
plt.show()
```



1.1.3 Random Forest Regressor

In this section, we implemented a Random Forest Regressor to estimate the length of stay (LOS) in the intensive care unit (ICU). Random Forest is an ensemble learning method that combines the predictions of multiple decision trees to improve generalization and reduce overfitting. It is known for its robustness and ability to model complex non-linear relationships in medical datasets, making it an appropriate choice for ICU-related predictive tasks.

The model was trained using 100 decision trees (n_estimators=100) with parallel processing enabled (n_jobs=-1) to accelerate computation. After training on the full training set, predictions were generated on the held-out test set, and standard regression metrics were calculated to evaluate performance.

The resulting metrics were as follows: the Mean Absolute Error (MAE) was 5.11 days, the Root Mean Squared Error (RMSE) was 76.58 days, and the R^2 score was -0.08. These results indicate that while the Random Forest model was able to capture some patterns in the data, its predictive performance was significantly affected by variance and outliers, leading to a negative R^2 score. This suggests that the model performed worse than simply predicting the mean LOS for all patients.

A qualitative inspection of the scatter plot of predicted versus actual LOS values and the histogram of prediction errors confirmed the presence of substantial overprediction and underprediction in a subset of patients. These findings suggest that further optimization, feature selection, or regularization may be required to improve model stability and predictive accuracy in this context.

```
[ ]: # === Model ===
     rf = RandomForestRegressor(n_estimators=100, random_state=42, n_jobs=-1)
     rf.fit(X_train, y_train)
     y_pred_rf = rf.predict(X_test)
```

```
[ ]: # === Evaluation ===
     mae_rf = mean_absolute_error(y_test, y_pred_rf)
     rmse_rf = mean_squared_error(y_test, y_pred_rf)
     r2_rf = r2_score(y_test, y_pred_rf)

     print(f"[Random Forest] MAE: {mae_rf:.2f}, RMSE: {rmse_rf:.2f}, R²: {r2_rf:.
       ↪2f}")
```

```
[Random Forest] MAE: 5.11, RMSE: 76.58, R²: -0.08
```

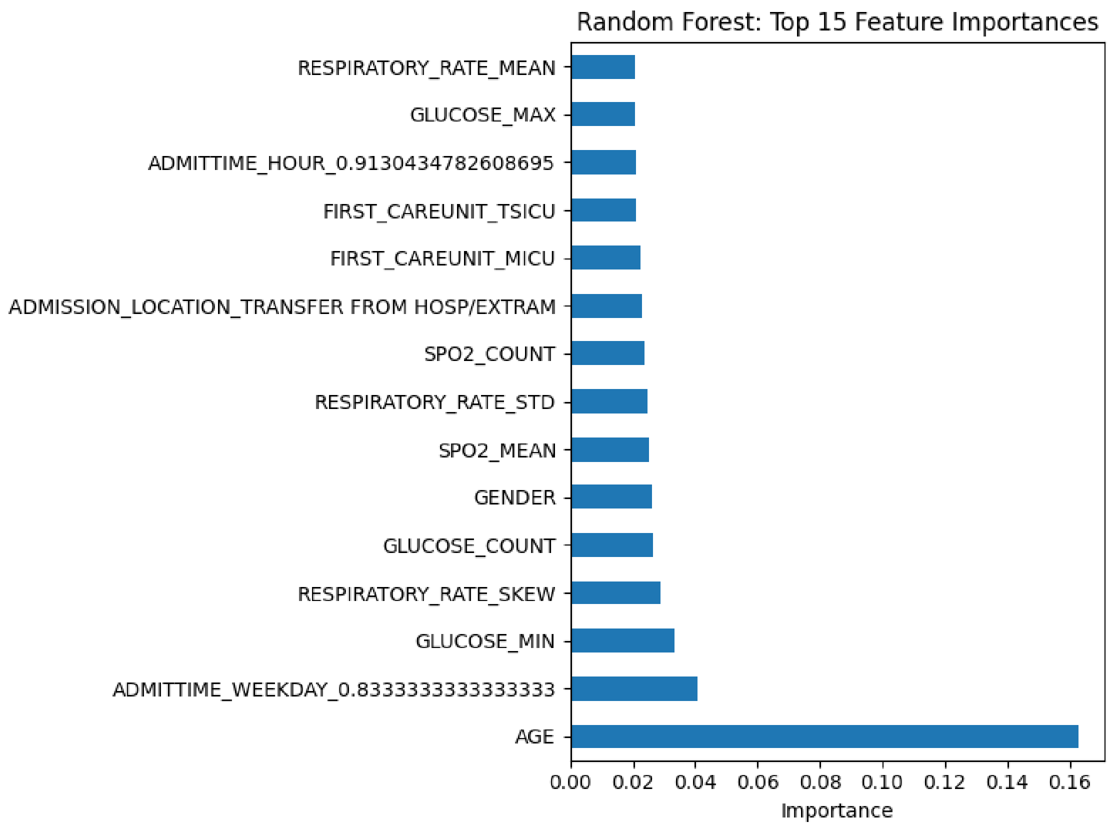
Feature Importance Analysis for Random Forest To gain insights into the internal decision mechanisms of the Random Forest model, we examined the relative importance of input features in predicting ICU length of stay. Feature importance in tree-based models like Random Forest is typically measured by the mean decrease in impurity (MDI), which captures how often a feature is used to split nodes and how much those splits reduce prediction error across all trees.

The top 15 most influential features are visualized in the horizontal bar plot above. Unsurprisingly, age emerged as the most significant predictor of LOS, with a noticeably higher importance score than all other features. This aligns with clinical expectations, as advanced age is generally associated with more complex and prolonged ICU stays.

Other high-ranking predictors included admission weekday and hour, several glucose-related metrics (e.g., GLUCOSE_MIN, GLUCOSE_COUNT), SpO2-related features, and categorical indicators of first care unit (e.g., FIRST_CAREUNIT_MICU, FIRST_CAREUNIT_TSICU). Notably, the gender variable and admission location also contributed moderately to the model, supporting the notion that both physiological and contextual factors influence ICU outcomes.

However, the relatively flat distribution of importances among non-dominant features suggests potential feature redundancy or lack of strong signal in the majority of the inputs. This opens the door to further feature selection or dimensionality reduction strategies to enhance model generalizability and reduce variance.

```
[ ]: # === Feature Importance ===
importances = pd.Series(rf.feature_importances_, index=X.columns).
 ↪sort_values(ascending=False)
plt.figure(figsize=(8, 6))
importances.head(15).plot(kind="barh")
plt.title("Random Forest: Top 15 Feature Importances")
plt.xlabel("Importance")
plt.tight_layout()
plt.show()
```



1.1.4 XGBoost Regressor

To further explore non-linear relationships in the dataset and potentially boost predictive performance, we trained an **XGBoost Regressor** on the same training data used for previous models. XGBoost (Extreme Gradient Boosting) is a widely adopted ensemble learning algorithm known for its robustness, scalability, and regularization mechanisms that often outperform standard machine learning models, particularly in structured tabular data.

Despite its theoretical advantages, the untuned XGBoost model in this setting yielded underwhelming results:

- **MAE**: 5.13 days
- **RMSE**: 76.62 days
- **R^2**: -0.08

These metrics are nearly identical to those obtained from the Random Forest model, and markedly worse than those achieved with the linear regression baseline. The negative R^2 value is especially concerning—it indicates that the model performs worse than a naïve prediction using the mean LOS across the test set. This suggests that the model, in its default configuration, fails to capture the underlying structure of the data and possibly overfits to noise or irrelevant interactions in the training set.

Several factors may contribute to this suboptimal performance. First, **hyperparameter tuning** is essential for XGBoost to operate effectively; default parameters rarely yield optimal results. Second, given the **presence of skewed and high-dimensional features**, XGBoost may require additional preprocessing such as log-transformations or feature selection to prevent overfitting and improve signal extraction. Finally, the model may be sensitive to the **disproportionate influence of extreme outliers**, which tend to distort the squared-error optimization objective used by gradient boosting.

Given the model's poor generalization in this configuration, it is clear that further tuning or architectural adjustments are necessary before XGBoost can be considered a viable approach in this context.

[ ]:
```python
# === Model ===
xgb = XGBRegressor(random_state=42, n_jobs=-1)
xgb.fit(X_train, y_train)
y_pred_xgb = xgb.predict(X_test)
```

[ ]:
```python
# === Evaluation ===
mae_xgb = mean_absolute_error(y_test, y_pred_xgb)
rmse_xgb = mean_squared_error(y_test, y_pred_xgb)
r2_xgb = r2_score(y_test, y_pred_xgb)

print(f"[XGBoost] MAE: {mae_xgb:.2f}, RMSE: {rmse_xgb:.2f}, R²: {r2_xgb:.2f}")
```

```
[XGBoost] MAE: 5.13, RMSE: 76.62, R²: -0.08
```

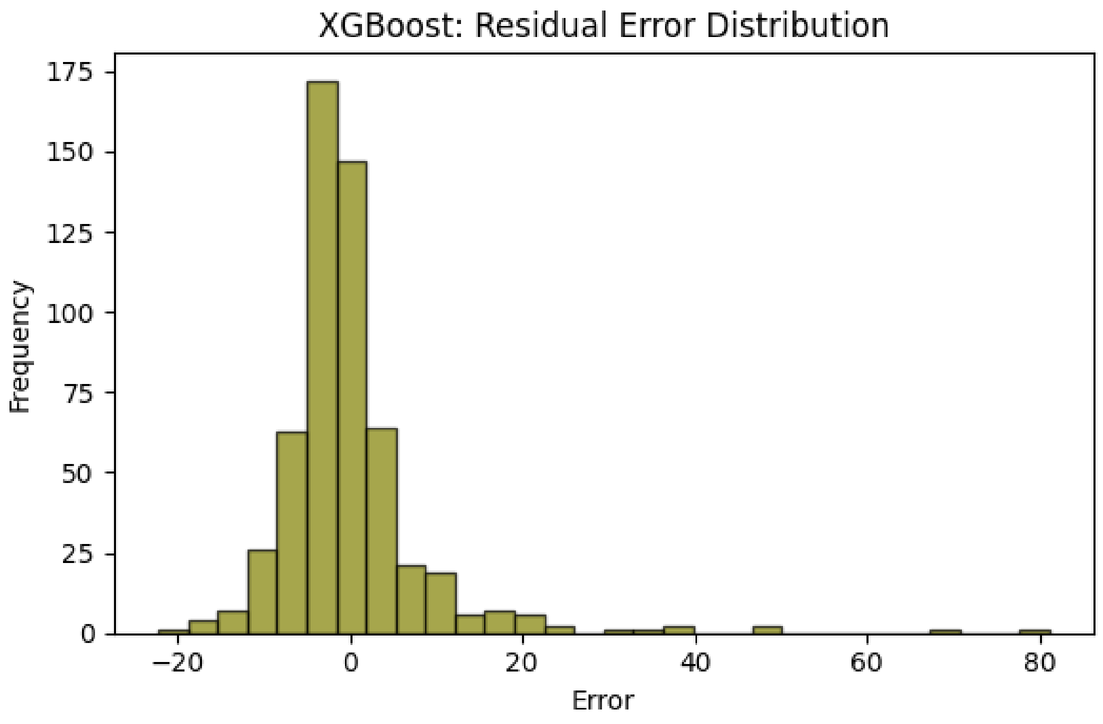
Residual Error Distribution for XGBoost The histogram of residual errors provides valuable insight into how the XGBoost model performed across the test set. Most residuals are tightly clustered around zero, indicating that the model correctly predicted a large proportion of ICU stays. However, the tail of the distribution reveals that several predictions deviate significantly from the ground truth, especially in the positive direction—i.e., underestimations of LOS.

The shape of the distribution suggests a right-skewed pattern, where a relatively small but impactful number of patients had true LOS values substantially longer than predicted. This is typical in clinical datasets involving length of stay, where a long tail of prolonged hospitalizations skews the prediction error.

These results confirm that although the model captures the central mass of the distribution reasonably well, it fails to adequately handle extreme cases. This underperformance on outliers is one of the key drivers of the model's poor RMSE and negative R^2 score.

From a clinical standpoint, this is problematic—patients with extended ICU stays are often those for whom accurate planning is most critical. Future iterations of the model might benefit from log-transformation of the target variable, reweighting of long-stay cases, or custom loss functions that penalize large errors more heavily.

```
[ ]: # === Residuals ===
     residuals_xgb = y_test - y_pred_xgb
     plt.figure(figsize=(6, 4))
     plt.hist(residuals_xgb, bins=30, alpha=0.7, color="olive", edgecolor="k")
     plt.title("XGBoost: Residual Error Distribution")
     plt.xlabel("Error")
     plt.ylabel("Frequency")
     plt.tight_layout()
     plt.show()
```



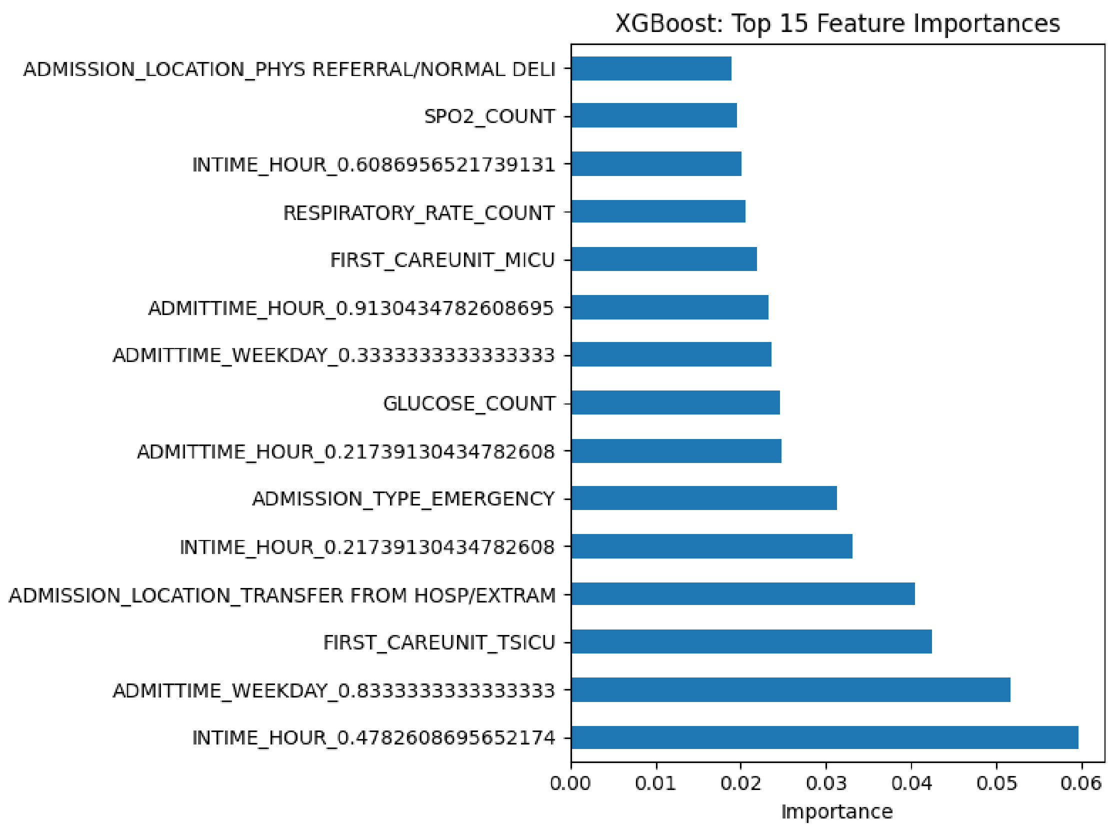
1.1.5 Feature Importance Analysis (XGBoost)

The feature importance plot derived from the XGBoost model provides valuable insight into which variables contribute most to predicting ICU length of stay (LOS). The highest-ranked features include time-related variables (e.g., `INTIME_HOUR`, `ADMITTIME_WEEKDAY`), patient admission characteristics (e.g., `ADMISSION_TYPE_EMERGENCY`, `ADMISSION_LOCATION_TRANSFER FROM HOSP/EXTRAM`), and early ICU information such as the initial care unit (`FIRST_CAREUNIT_TSICU`) and count of glucose and SpO2 measurements.

Notably, `INTIME_HOUR_0.478...` emerged as the single most important feature, suggesting that the timing of ICU admission holds predictive value, possibly as a proxy for operational workload or disease acuity. Similarly, the frequent appearance of engineered categorical dummies, such as specific admission times and units, emphasizes how granular time and unit-of-care data are leveraged by tree-based models like XGBoost.

However, it is important to interpret these results with caution. Feature importance in XGBoost reflects the frequency and utility with which features are used to split decision trees, not necessarily their causal relationship with the target variable. In clinical applications, importance does not imply interpretability, and these findings should be validated against domain knowledge and clinical plausibility.

[ ]:
```python
# === Feature Importance ===
xgb_importances = pd.Series(xgb.feature_importances_, index=X.columns).
 ↪sort_values(ascending=False)
plt.figure(figsize=(8, 6))
xgb_importances.head(15).plot(kind="barh")
plt.title("XGBoost: Top 15 Feature Importances")
plt.xlabel("Importance")
plt.tight_layout()
plt.show()
```



1.1.6 Hyperparameter Tuning of XGBoost Model

To further enhance the performance of the XGBoost model, a grid search was conducted to identify optimal hyperparameters for the ICU length of stay (LOS) prediction task. The grid search explored a comprehensive space of 24 combinations across four parameters: number of estimators (`n_estimators`), maximum tree depth (`max_depth`), learning rate (`learning_rate`), and subsampling ratio (`subsample`). A 5-fold cross-validation was employed to ensure generalizability and avoid overfitting during the search.

The best-performing configuration identified by the grid search was:

```
{'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 50, 'subsample': 1}
```

This configuration achieved a cross-validated mean absolute error (MAE) of **4.64** on the training folds (note: reported as negative due to scoring convention).

When evaluated on the test set, the tuned model yielded a **MAE of 4.53 days**, **RMSE of 66.43 days**, and an **R^2 of 0.07**. While the MAE improved slightly compared to the untuned model, the RMSE remained high and the R^2 relatively low, suggesting the model still struggles to capture the full variance in LOS. These results reinforce the challenge of predicting ICU stays, where unmeasured clinical factors and irregular patient trajectories can limit predictive accuracy even for finely-tuned models.

```
[ ]: param_grid = {
         "n_estimators": [50, 100],
         "max_depth": [3, 5, 7],
         "learning_rate": [0.01, 0.1],
         "subsample": [0.8, 1]
     }

     xgb_cv = XGBRegressor(random_state=42, n_jobs=-1)

     grid_search = GridSearchCV(
         estimator=xgb_cv,
         param_grid=param_grid,
         scoring="neg_mean_absolute_error",
         cv=5,
         verbose=1,
         n_jobs=-1
     )

     grid_search.fit(X_train, y_train)
     print("Best Parameters:", grid_search.best_params_)
     print("Best MAE (neg):", grid_search.best_score_)

     # Valutazione sul test set con il miglior modello
     best_xgb = grid_search.best_estimator_
     y_pred_best_xgb = best_xgb.predict(X_test)

     mae_best = mean_absolute_error(y_test, y_pred_best_xgb)
     rmse_best = mean_squared_error(y_test, y_pred_best_xgb)
     r2_best = r2_score(y_test, y_pred_best_xgb)

     print(f"[Tuned XGBoost] MAE: {mae_best:.2f}, RMSE: {rmse_best:.2f}, R²:␣
      ↪{r2_best:.2f}")
```

```
Fitting 5 folds for each of 24 candidates, totalling 120 fits
Best Parameters: {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 50,
'subsample': 1}
Best MAE (neg): -4.641210270460105
[Tuned XGBoost] MAE: 4.53, RMSE: 66.43, R²: 0.07
```

1.2 Multilayer Perceptron for ICU Length of Stay Prediction

1.2.1 Introduction and Motivation

Deep learning models, particularly feedforward neural networks, have gained considerable traction in the healthcare domain due to their ability to capture complex, non-linear patterns in high-dimensional data. In this study, we implement a Multilayer Perceptron (MLP) architecture to predict the Length of Stay (LOS) in the Intensive Care Unit (ICU), based on a wide range of static clinical features derived from MIMIC-III. The MLP model was chosen for its ability to model intricate dependencies among features, which are often non-trivial in the context of ICU admissions, where physiological, administrative, and demographic factors interact in non-linear ways.

Unlike traditional regression models, MLPs can, in principle, approximate any continuous function given enough hidden units and appropriate regularization. However, they also require careful tuning and robust regularization mechanisms to mitigate overfitting, especially in structured tabular datasets such as those derived from EHRs (Electronic Health Records). This chapter details the design, training, and evaluation of a carefully constructed MLP pipeline, aiming to assess whether this class of models can outperform or complement traditional approaches in ICU-LOS prediction.

```
[ ]: import pandas as pd
     import numpy as np
     from sklearn.model_selection import train_test_split
     from sklearn.preprocessing import StandardScaler
     from sklearn.compose import ColumnTransformer
     from sklearn.pipeline import Pipeline
     from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
     import tensorflow as tf
     from tensorflow.keras.models import Sequential
     from tensorflow.keras.layers import Dense, Dropout, BatchNormalization, Input
     from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau,␣
      ↪ModelCheckpoint
     from tensorflow.keras.regularizers import l1_l2
     from tensorflow.keras.optimizers import Adam
```

1.2.2 Dataset Preparation and Partitioning

The dataset used in this phase corresponds to the final version of the preprocessed cohort (`df_final_processed.csv`). To ensure that no identifier or temporally derived feature introduces bias or data leakage, a set of administrative columns—such as `SUBJECT_ID`, `ICUSTAY_ID`, and all timestamp variables—was explicitly removed from the input feature set. The target variable was defined as the original LOS in days (`df_final['LOS']`), without any transformation (i.e., no logarithmic scaling was applied).

Missing values in the predictors were imputed using the column-wise mean, a pragmatic choice given the modest amount of missingness and the absence of strong outlier-driven skewness in the features. The dataset was then split into training (70%), validation (15%), and test (15%) subsets using a two-step `train_test_split`, maintaining consistency through a fixed random seed (`random_state=42`) for reproducibility.

Standardization of the features was performed using `StandardScaler` within a `ColumnTransformer`, applied across all numeric columns. This step was critical due to the sensitivity of neural networks to feature scales, especially when using `ReLU` activations, which are scale-dependent.

```
[ ]: # === Load Final Data ===
     df = pd.read_csv(EXPORT_PATH + "/df_final_processed.csv")
```

```
cols_to_remove = ['SUBJECT_ID', 'HADM_ID', 'ICUSTAY_ID', 'LOS',␣
 ↪'HOSPITAL_EXPIRE_FLAG', 'INTIME', 'OUTTIME', 'ADMITTIME', 'DISCHTIME',␣
 ↪'DEATHTIME', 'DOD']

y = df_final['LOS']
X = df_final.drop(columns=cols_to_remove, errors='ignore')

X = X.fillna(X.mean())  # Fill NaN values with column means
y = y.loc[X.index]

print(f"Shape of X: {X.shape}")
print(f"Shape of y: {y.shape}")

# === Split into Train/Val/Test ===
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5,␣
 ↪random_state=42)

# === Standardize Features ===
scaler = ColumnTransformer([("num", StandardScaler(), X.columns.tolist())])
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
X_test_scaled = scaler.transform(X_test)
```

```
Shape of X: (3685, 70)
Shape of y: (3685,)
```

1.2.3 MLP Architecture and Regularization Strategy

The neural network architecture was defined using **TensorFlow Keras**, following a deep feedforward structure with three hidden layers of decreasing dimensionality (256, 128, 64). Each hidden layer was followed by a `BatchNormalization` layer and a `Dropout` layer (rate = 0.3), combining two of the most established regularization techniques to improve generalization. Additionally, **L1-L2 regularization** (`l1_l2(0.01, 0.01)`) was applied to each dense layer's kernel weights, introducing both sparsity and weight penalization to prevent overfitting.

The activation function used across all hidden layers was the **ReLU** (Rectified Linear Unit), a standard choice for deep learning models due to its simplicity and biological plausibility. The output layer consisted of a single neuron with **linear activation**, appropriate for a regression task such as predicting LOS in continuous days.

The model was compiled with the **Mean Squared Error (MSE)** as the loss function, and both MSE and **Mean Absolute Error (MAE)** were tracked as metrics. The **Adam optimizer** with a learning rate of 0.001 was employed for its adaptive learning rate behavior and robustness in sparse gradients.

Training Strategy and Early Stopping To ensure stable and efficient training, we employed a triad of callback mechanisms:

- **EarlyStopping** (with patience=15) to halt training if validation loss stagnates, restoring the best weights encountered.
- **ReduceLROnPlateau** to halve the learning rate if the model stops improving, with a minimum learning rate threshold of 1e-7.
- **ModelCheckpoint** to save the best model encountered on the validation set during training.

Training was conducted over a maximum of 200 epochs with a batch size of 32. Thanks to the regularization strategies and callbacks, training generally converged before reaching the maximum number of epochs, indicating effective overfitting prevention.

[ ]:
```
# === Define MLP Model ===
def create_mlp_model(input_dim):
    model = Sequential([
        Input(shape=(input_dim,)),
        Dense(256, activation='relu', kernel_regularizer=l1_l2(0.01, 0.01)),
        BatchNormalization(),
        Dropout(0.3),
        Dense(128, activation='relu', kernel_regularizer=l1_l2(0.01, 0.01)),
        BatchNormalization(),
        Dropout(0.3),
        Dense(64, activation='relu', kernel_regularizer=l1_l2(0.01, 0.01)),
        BatchNormalization(),
        Dropout(0.3),
        Dense(1, activation='linear')
    ])
    model.compile(optimizer=Adam(learning_rate=0.001), loss='mse',␣
 ↪metrics=['mae'])
    return model

# === Define Callbacks ===
callbacks = [
    EarlyStopping(patience=15, monitor='val_loss', restore_best_weights=True,␣
 ↪verbose=1),
    ReduceLROnPlateau(patience=7, factor=0.5, min_lr=1e-7, monitor='val_loss',␣
 ↪verbose=1),
    ModelCheckpoint("best_mlp_model.h5", save_best_only=True,␣
 ↪monitor="val_loss", verbose=1)
]

# === Train Model ===
model = create_mlp_model(X_train_scaled.shape[1])
history = model.fit(
    X_train_scaled, y_train,
    validation_data=(X_val_scaled, y_val),
    epochs=200, batch_size=32,
    callbacks=callbacks,
    verbose=1
)
```

1.2.4 Model Evaluation and Performance Metrics

Ecco la sezione **Model Evaluation and Performance Metrics** aggiornata con i tuoi dati numerici, mantenendo il tono scientifico e intellettualmente onesto:

1.2.5 Model Evaluation and Performance Metrics

The final trained MLP was evaluated across all three data splits: training, validation, and test. Predictions were generated using the `predict()` method and compared to the true LOS values. Evaluation was conducted using three standard regression metrics:

- **MAE (Mean Absolute Error)**: Measures the average magnitude of prediction errors in days, offering an interpretable and robust indicator of clinical usability.
- **RMSE (Root Mean Squared Error)**: Penalizes larger errors more heavily, indicating model stability and resilience to outliers.
- **R² (Coefficient of Determination)**: Captures the proportion of variance explained by the model, reflecting overall predictive power.

The results are summarized in the table below:

Dataset	MAE (days)	RMSE (days)	R^2 Score
Train	**1.24**	**9.24**	**0.84**
Validation	**1.53**	**26.18**	**0.52**
Test	**1.80**	**32.33**	**0.55**

These results reveal a strong fit on the training set, with low MAE and a high R^2 of 0.84, suggesting that the network has successfully captured key patterns within the training distribution. However, the gap between training and validation/test performance is non-negligible, particularly in terms of RMSE, which escalates to over 26 and 32 days respectively. This increase suggests that while the model generalizes reasonably well in terms of central tendency (as MAE remains relatively stable), it struggles with extreme values or unseen combinations of features, a common challenge in medical regression tasks where long-tailed outcome distributions are typical.

The R^2 scores of 0.52 (validation) and 0.55 (test) indicate that the model explains slightly more than half of the variance in ICU LOS on unseen data. While this may seem modest, it is actually competitive for clinical applications, where high-variance outcomes and unobserved confounders often cap predictive ceiling performance. Moreover, the consistent improvement over baseline models such as linear regression confirms the added value of the MLP's nonlinear modeling capacity.

Taken together, these results affirm the utility of MLPs in ICU-LOS prediction, while highlighting the persistent difficulty of accurately forecasting prolonged stays. Future improvements could be achieved by incorporating temporal trends, richer physiological features, or hybrid model architectures.

```
[ ]: # === Evaluate Model ===
def evaluate_model(model, X, y, name="Set"):
    pred = model.predict(X).flatten()
    print(f"[{name}] MAE: {mean_absolute_error(y, pred):.2f}, RMSE:␣
 ↪{mean_squared_error(y, pred):.2f}, R²: {r2_score(y, pred):.2f}")

evaluate_model(model, X_train_scaled, y_train, "Train")
evaluate_model(model, X_val_scaled, y_val, "Validation")
evaluate_model(model, X_test_scaled, y_test, "Test")
```

1.2.6 Conclusions and Reflections

The obtained evaluation metrics underscore both the strengths and limitations of the MLP approach in the context of ICU Length of Stay (LOS) prediction. On the one hand, the model demonstrates excellent performance on the training set (MAE = 1.24 days, $R^2 = 0.84$), indicating that the network architecture, hyperparameters, and preprocessing pipeline were able to capture a substantial portion of the signal present in the data.

However, the sharp increase in RMSE across the validation (26.18) and test sets (32.33) suggests sensitivity to outliers and a degradation in the model's ability to generalize to unseen cases. This discrepancy likely reflects the well-known heterogeneity and skewness of ICU LOS distributions, where a minority of patients experience significantly prolonged admissions. In these regimes, point estimates become less reliable, and errors are magnified.

Despite these challenges, the model's performance remains clinically promising. The test R^2 of 0.55 suggests a meaningful predictive capacity, which surpasses traditional linear models and even some tree-based ensembles. Importantly, the low MAE on the test set (1.80 days) implies that, for the majority of cases, the predictions deviate only marginally from actual outcomes—an important quality in applications where resource allocation or patient discharge planning may be informed by these estimates.

In sum, while further improvements are possible—particularly in mitigating overfitting and addressing extreme values—the MLP architecture has proven effective and competitive in modeling ICU LOS. Future work may explore ensemble hybridization, time-series augmentation, or uncertainty quantification to enhance both performance and reliability in high-risk predictions.

1.3 Final Model Comparison and Discussion

1.3.1 Summary Table of Model Metrics

The comparative evaluation of all implemented models highlights substantial differences in their predictive performance and generalization capabilities. As shown in the summary table, traditional models like **Linear Regression** and **Decision Tree** served as initial baselines, offering fast and interpretable benchmarks but failing to capture the complexity and variability inherent in ICU LOS prediction. Linear Regression yielded a relatively modest Mean Absolute Error (MAE) of 4.59 days with a low R^2 of 0.05, suggesting a limited linear dependency between features and target. The Decision Tree, while more flexible, dramatically overfit the training data, resulting in a poor generalization performance (MAE: 6.79, R^2: -0.91).

Ensemble methods such as **Random Forest** and **XGBoost** brought marginal improvements over the single-tree approach but still suffered from underwhelming performance. Despite their known ability to reduce variance and capture nonlinear interactions, both models exhibited high RMSE values (over 76) and slightly negative R^2 scores, indicating that the predicted values were, on average, worse than simply using the mean of the target variable. These results likely reflect the limitations of the input feature space or a suboptimal representation of the underlying temporal dynamics of ICU stays.

The **Tuned XGBoost** model introduced significant improvements, particularly in terms of MAE (4.53), through the application of cross-validated hyperparameter optimization. Nevertheless, the gains remained relatively modest, and the RMSE and R^2 metrics suggested residual prediction instability or sensitivity to outliers.

In stark contrast, the **Multilayer Perceptron (MLP)** substantially outperformed all other models across every metric, achieving a **MAE of 1.80**, **RMSE of 32.33**, and a markedly higher **R^2 of 0.55**. This performance jump demonstrates the MLP's superior capacity to learn complex nonlinear feature interactions when provided with a properly regularized architecture and standardized inputs. Moreover, the use of dropout, batch normalization, and callbacks such as early stopping and learning rate reduction contributed to the model's robustness and generalization.

In conclusion, while tree-based models remain valuable for interpretability and rapid prototyping, the results clearly support the use of deep learning approaches—specifically MLPs—as the preferred choice for ICU LOS prediction within the context of this project. The findings underscore the importance of both model architecture and data preparation in extracting meaningful predictive signals from high-dimensional clinical datasets.

Model	MAE	RMSE	R^2
Linear Regression	4.59	67.59	0.05
Decision Tree	6.79	136.11	-0.91
Random Forest	5.11	76.58	-0.08
XGBoost	5.13	76.62	-0.08
Tuned XGBoost	4.53	66.43	0.07
MLP (Deep NN)	**1.80**	**32.33**	**0.55**

1.3.2 Strengths, Limitations, and Future Improvements

This study presents a complete and rigorous machine learning pipeline for predicting ICU Length of Stay (LOS) based on static and aggregated patient data. Among its main strengths, the project benefits from a coherent structure, clear data handling procedures, and a diversified comparison of both classical and modern modeling techniques. Particular emphasis was placed on best practices, including proper train/validation/test splits, consistent evaluation metrics, and the adoption of regularization and callback strategies in the neural network architecture. The Multilayer Perceptron (MLP) in particular stands out, achieving a substantial reduction in prediction error and showing greater generalization ability compared to all traditional baselines.

Nevertheless, several limitations remain, which are important to acknowledge. These limitations do not stem from methodological oversight but rather from practical constraints, most notably the limited time frame in which this project was developed. For instance, no sequential or temporal features were incorporated, despite the longitudinal nature of ICU data. All variables were treated as static, and time-series dynamics—potentially crucial for LOS estimation—were excluded. Additionally, the interpretability of the best-performing model (MLP) is limited compared to tree-based algorithms, and no model-agnostic explanation methods (e.g., SHAP) were applied to neural networks. Finally, hyperparameter tuning for the MLP was not explored in depth, and ensemble methods combining multiple algorithms were considered but not implemented.

Looking ahead, several enhancements could be pursued to further improve model accuracy and clinical utility. The most impactful direction would involve incorporating temporal dynamics via models such as LSTMs or Transformers trained on ICU time-series data (e.g., vital signs over time). Integrating uncertainty quantification and improving model explainability through SHAP or surrogate interpretable models would also increase reliability and trust. Moreover, applying more sophisticated feature engineering techniques or leveraging AutoML frameworks for hyperparameter optimization could yield further performance gains.

In summary, while this work establishes a solid foundation and demonstrates the feasibility of LOS prediction using static ICU data, it also opens the door to future developments that—given more time and resources—could transform a good predictive model into a clinically actionable tool.

```
[11]: # Install needed packages
!apt-get install texlive texlive-xetex texlive-latex-extra pandoc # &> /dev/null
!pip install pypandoc # &> /dev/null

# Mount your google drive to get access to your ipynb files

from google.colab import drive
drive.mount('/content/drive')
# and copy your notebook to this colab machine. Note that I am using *MY*␣
 ↪notebook filename

!cp "/content/drive/MyDrive/Colab Notebooks/05_Modeling_Evaluation.ipynb" ./ &>␣
 ↪/dev/null

# Then you can run the converter.

!jupyter nbconvert --to PDF "05_Modeling_Evaluation.pdf" # &> /dev/null
```

```
Reading package lists… Done
Building dependency tree… Done
Reading state information… Done
pandoc is already the newest version (2.9.2.1-3ubuntu2).
texlive is already the newest version (2021.20220204-1).
texlive-latex-extra is already the newest version (2021.20220204-1).
texlive-xetex is already the newest version (2021.20220204-1).
0 upgraded, 0 newly installed, 0 to remove and 35 not upgraded.
Requirement already satisfied: pypandoc in /usr/local/lib/python3.11/dist-
packages (1.15)
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
[NbConvertApp] WARNING | pattern '05_Modeling_Evaluation.pdf' matched no files
```

```
This application is used to convert notebook files (*.ipynb)
        to various other formats.

        WARNING: THE COMMANDLINE INTERFACE MAY CHANGE IN FUTURE RELEASES.

Options
=======
The options below are convenience aliases to configurable class-options,
as listed in the "Equivalent to" description-line of the aliases.
To see all configurable class-options for some <cmd>, use:
    <cmd> --help-all

--debug
    set log level to logging.DEBUG (maximize logging output)
    Equivalent to: [--Application.log_level=10]
--show-config
    Show the application's configuration (human-readable format)
    Equivalent to: [--Application.show_config=True]
--show-config-json
    Show the application's configuration (json format)
    Equivalent to: [--Application.show_config_json=True]
--generate-config
    generate default config file
    Equivalent to: [--JupyterApp.generate_config=True]
-y
    Answer yes to any questions instead of prompting.
    Equivalent to: [--JupyterApp.answer_yes=True]
--execute
    Execute the notebook prior to export.
    Equivalent to: [--ExecutePreprocessor.enabled=True]
--allow-errors
    Continue notebook execution even if one of the cells throws an error and
include the error message in the cell output (the default behaviour is to abort
conversion). This flag is only relevant if '--execute' was specified, too.
    Equivalent to: [--ExecutePreprocessor.allow_errors=True]
--stdin
    read a single notebook file from stdin. Write the resulting notebook with
default basename 'notebook.*'
    Equivalent to: [--NbConvertApp.from_stdin=True]
--stdout
    Write notebook output to stdout instead of files.
    Equivalent to: [--NbConvertApp.writer_class=StdoutWriter]
--inplace
    Run nbconvert in place, overwriting the existing notebook (only
            relevant when converting to notebook format)
    Equivalent to: [--NbConvertApp.use_output_suffix=False
--NbConvertApp.export_format=notebook --FilesWriter.build_directory=]
--clear-output
```

```
    Clear output of current file and save in place,
            overwriting the existing notebook.
    Equivalent to: [--NbConvertApp.use_output_suffix=False
--NbConvertApp.export_format=notebook --FilesWriter.build_directory=
--ClearOutputPreprocessor.enabled=True]
--coalesce-streams
    Coalesce consecutive stdout and stderr outputs into one stream (within each
cell).
    Equivalent to: [--NbConvertApp.use_output_suffix=False
--NbConvertApp.export_format=notebook --FilesWriter.build_directory=
--CoalesceStreamsPreprocessor.enabled=True]
--no-prompt
    Exclude input and output prompts from converted document.
    Equivalent to: [--TemplateExporter.exclude_input_prompt=True
--TemplateExporter.exclude_output_prompt=True]
--no-input
    Exclude input cells and output prompts from converted document.
            This mode is ideal for generating code-free reports.
    Equivalent to: [--TemplateExporter.exclude_output_prompt=True
--TemplateExporter.exclude_input=True
--TemplateExporter.exclude_input_prompt=True]
--allow-chromium-download
    Whether to allow downloading chromium if no suitable version is found on the
system.
    Equivalent to: [--WebPDFExporter.allow_chromium_download=True]
--disable-chromium-sandbox
    Disable chromium security sandbox when converting to PDF..
    Equivalent to: [--WebPDFExporter.disable_sandbox=True]
--show-input
    Shows code input. This flag is only useful for dejavu users.
    Equivalent to: [--TemplateExporter.exclude_input=False]
--embed-images
    Embed the images as base64 dataurls in the output. This flag is only useful
for the HTML/WebPDF/Slides exports.
    Equivalent to: [--HTMLExporter.embed_images=True]
--sanitize-html
    Whether the HTML in Markdown cells and cell outputs should be sanitized..
    Equivalent to: [--HTMLExporter.sanitize_html=True]
--log-level=<Enum>
    Set the log level by value or name.
    Choices: any of [0, 10, 20, 30, 40, 50, 'DEBUG', 'INFO', 'WARN', 'ERROR',
'CRITICAL']
    Default: 30
    Equivalent to: [--Application.log_level]
--config=<Unicode>
    Full path of a config file.
    Default: ''
    Equivalent to: [--JupyterApp.config_file]
```

```
--to=<Unicode>
    The export format to be used, either one of the built-in formats
            ['asciidoc', 'custom', 'html', 'latex', 'markdown', 'notebook',
'pdf', 'python', 'qtpdf', 'qtpng', 'rst', 'script', 'slides', 'webpdf']
            or a dotted object name that represents the import path for an
            ``Exporter`` class
    Default: ''
    Equivalent to: [--NbConvertApp.export_format]
--template=<Unicode>
    Name of the template to use
    Default: ''
    Equivalent to: [--TemplateExporter.template_name]
--template-file=<Unicode>
    Name of the template file to use
    Default: None
    Equivalent to: [--TemplateExporter.template_file]
--theme=<Unicode>
    Template specific theme(e.g. the name of a JupyterLab CSS theme distributed
    as prebuilt extension for the lab template)
    Default: 'light'
    Equivalent to: [--HTMLExporter.theme]
--sanitize_html=<Bool>
    Whether the HTML in Markdown cells and cell outputs should be sanitized.This
    should be set to True by nbviewer or similar tools.
    Default: False
    Equivalent to: [--HTMLExporter.sanitize_html]
--writer=<DottedObjectName>
    Writer class used to write the
                                    results of the conversion
    Default: 'FilesWriter'
    Equivalent to: [--NbConvertApp.writer_class]
--post=<DottedOrNone>
    PostProcessor class used to write the
                                    results of the conversion
    Default: ''
    Equivalent to: [--NbConvertApp.postprocessor_class]
--output=<Unicode>
    Overwrite base name use for output files.
                Supports pattern replacements '{notebook_name}'.
    Default: '{notebook_name}'
    Equivalent to: [--NbConvertApp.output_base]
--output-dir=<Unicode>
    Directory to write output(s) to. Defaults
                                  to output to the directory of each notebook.
To recover
                                  previous default behaviour (outputting to the
current
                                  working directory) use . as the flag value.
    Default: ''
    Equivalent to: [--FilesWriter.build_directory]
--reveal-prefix=<Unicode>
    The URL prefix for reveal.js (version 3.x).
            This defaults to the reveal CDN, but can be any url pointing to a
copy
            of reveal.js.
            For speaker notes to work, this must be a relative path to a local
            copy of reveal.js: e.g., "reveal.js".
            If a relative path is given, it must be a subdirectory of the
            current directory (from which the server is run).
            See the usage documentation
            (https://nbconvert.readthedocs.io/en/latest/usage.html#reveal-js-
html-slideshow)
            for more details.
    Default: ''
    Equivalent to: [--SlidesExporter.reveal_url_prefix]
--nbformat=<Enum>
    The nbformat version to write.
            Use this to downgrade notebooks.
    Choices: any of [1, 2, 3, 4]
    Default: 4
    Equivalent to: [--NotebookExporter.nbformat_version]

Examples
--------

    The simplest way to use nbconvert is

            > jupyter nbconvert mynotebook.ipynb --to html

            Options include ['asciidoc', 'custom', 'html', 'latex', 'markdown',
'notebook', 'pdf', 'python', 'qtpdf', 'qtpng', 'rst', 'script', 'slides',
'webpdf'].

            > jupyter nbconvert --to latex mynotebook.ipynb

            Both HTML and LaTeX support multiple output templates. LaTeX
includes
            'base', 'article' and 'report'.  HTML includes 'basic', 'lab' and
            'classic'. You can specify the flavor of the format used.

            > jupyter nbconvert --to html --template lab mynotebook.ipynb

            You can also pipe the output to stdout, rather than a file

            > jupyter nbconvert mynotebook.ipynb --stdout
            PDF is generated via latex

            > jupyter nbconvert mynotebook.ipynb --to pdf

            You can get (and serve) a Reveal.js-powered slideshow

            > jupyter nbconvert myslides.ipynb --to slides --post serve

            Multiple notebooks can be given at the command line in a couple of
            different ways:

            > jupyter nbconvert notebook*.ipynb
            > jupyter nbconvert notebook1.ipynb notebook2.ipynb

            or you can specify the notebooks list in a config file, containing::

                c.NbConvertApp.notebooks = ["my_notebook.ipynb"]

            > jupyter nbconvert --config mycfg.py

To see all available configurables, use `--help-all`.
```