# Predicting Hotel Booking Cancellations and Average Daily Rate (ADR) for Optimized Revenue Management

Sainath Aittla
*SEAS*
*University at Buffalo*
UBIT: 50557807
saittla@buffalo.edu

Santoshi Reddy Chintala
*SEAS*
*University at Buffalo*
UBIT: 50559623
santoshi@buffalo.edu

Pranavi Chintala
*SEAS*
*University at Buffalo*
UBIT: 50563997
pchintal@buffalo.edu

***Abstract*—The hospitality industry has to bear a lot of revenue optimization difficulties because of its fluctuating pattern of booking and very high cancellation rates. This research will analyze hotel reservation data with the aim of identifying the most influential factors governing cancellations in bookings, and forecasting the average daily rate, a critical performance metric defined as the ratio of total lodging transactions to the total number of nights stayed. The dataset used for this study is from Kaggle, containing 119,390 records across 32 attributes, reflecting the whole gamut of booking trends, demographics of guests, and status of reservations. We go through a lot of data cleaning and processing: handling missing values, removing columns because many are irrelevant, and converting data types in order to prepare this dataset for predictive modeling and analysis. This research utilizes Apache Spark for handling and processing large datasets efficiently, enabling scalable and distributed data analysis. The research will, therefore, be instrumental in guiding hotels to make use of dynamic pricing in improving customer experiences and maximizing occupancy by addressing questions to do with cancellations and ADR. Results from the project have a great possibility of considerably helping revenue management practices within the hospitality industry.**

## I. Introduction

The hospitality industry is among the most growth-oriented and competitive sectors around the world's economy. Hotels have to deal with market demand, consumer behavior changes, and seasonality to maintain profitability on a day-to-day basis. Two of the most critical factors in revenue management for a hotel include cancellations made on reservations and Average Daily Rate. A high level of cancellation rate can seriously disrupt the operations of a hotel and lead to huge losses in revenues and resource utilization mismatch. On the other hand, ADR is a major indicator of performance since it is a reflection of the hotel's power in charging and deriving revenue from selling the rooms. Comprehension and anticipation of these factors are very important in allowing hotels to develop right pricing techniques that ensure full occupancy and customers enjoy their stay.

The goals of this research include an investigation into the pattern exhibited by customers based on hotel booking data, identification of major drivers of cancellation, and the prediction of ADR. This will, in turn, help hotels make major informed decisions by reducing cancellations, optimizing prices, and enhancing the overall customer experience. In fact, ADR prediction is a key feature because it draws a picture of the financial health of a hotel. It empowers these companies to change their strategies in any given time, based on market conditions, seasonality, and demographics. The dataset for this study has been derived from Kaggle, containing 119,390 records with 32 attributes. This represents a very rich source of information, including types of hotels, booking dates, guest demographics, length of stay, and reservation status. This elongated dataset will enable them to explore exhaustive dimensions of factors that affect cancellations and ADR for strategic revenue management.

The data cleaning and different preprocessing steps get the dataset ready for analysis and modeling. Apache Spark is utilized in this study to handle and process the large dataset efficiently, providing distributed and scalable computation capabilities. This ensures faster and more efficient data processing, making it possible to analyze patterns and build predictive models on a large volume of data. We want to comprehend which characteristics in bookings are most closely linked with cancellations, what guest profile has the most impact, and how this influences ADR. This, in turn, shall provide practical insights for hotels to shape their operations, offer personalized marketing, and optimize profitability. This now helps hotels gain better insight into more pragmatic ways to reduce revenue loss due to cancellations and to optimize their pricing models towards the demands of customers and market dynamics.

## II. Problem Statement

The project will analyze hotel booking data to demonstrate trends in customer bookings, patterns, and leading causes of cancellations. It shall then focus on the ADR prediction, a critical financial metric in the hospitality industry that refers to the total sum of money earned from all lodging transactions divided by the total number of nights stayed. Since the hotel business is heavily dependent on ADR, the better

the prediction, the more it is able to help optimize revenue management and pricing strategies. Therefore, this study tries to answer some questions linked to the improvement of predictions. Which factors create cancellations? How effectively can we forecast ADR by using various booking and customer attributes? Which are the seasonal and market variations with respect to both hotel bookings and ADR? The project gives answers to all these questions so that hotels take positive steps toward customer satisfaction, improvement in revenue streams, and optimization of their operations. To support the analysis, distributed data preprocessing using PySpark has been employed. Extending the data preprocessing steps from Project Phase 1, the dataset was cleaned, transformed, and prepared for analysis using Resilient Distributed Dataset (RDD) operations and windowing techniques. This distributed approach ensures efficient handling of large datasets by leveraging parallel computing, thereby expediting the data preprocessing phase and enabling scalable data transformations necessary for effective modeling and analysis.

### *A. Background and Significance*

Revenue management is a significant factor in the success of hotels operating in today's highly competitive hospitality business environment. Hotels apply ADR as one of the main performance measures that provide an indication of their financial health. Variations in ADR also contribute to high cancellation ratios, thereby yielding massive shifts in revenue and operational efficiency among hotels. By doing this, the hotels will be able to compete in the market because, through this, they will have a deep understanding of the pattern of bookings, identification of key drivers leading to cancellations, and by forecasting ADR accurately, they can offer optimized pricing strategies. The addressal of such a problem is of great significance for several reasons:

It enables the hotel to manage revenue by dynamic pricing higher when demand is high and evening out the occupancy when demand for business is at a low.

Customer Retention Service: With the analysis of booking behavior and cancellation trends, hotels can come up with effective policies and promotions that will enhance customer satisfaction and loyalty.

Resource Allocation: Accurate ADRs will definitely ensure a proper utilization of resources, improving service and enhancing customer experience. In this respect, considering these factors, the contribution of the project can consist in the fact that with its help, data-driven insights will be able to provide hotels with an opportunity to elaborate on business strategies given the ever-changing market dynamics.

### *B. Contribution Potential*

The objective of this project is to give an added dimension to hotel revenue management through predictive analytics. By comprehending the causes of cancellation and ADR, hotels will be able to apply such knowledge in fine-tuning their price models and booking policies toward enhanced profitability and customer satisfaction. The predictions and insights derived from this study will be crucial for the following:

Dynamic Pricing: Development of models that recommend the optimal room rates based on the prediction made of ADR.

Cancellation Reduction: The process of identifying high-risk bookings and the application of targeted interventions to minimize cancellation rates.

Strategic Marketing: We can segment our customers by their booking habits and tailor promotions or offers accordingly.

In all, the outcome of this project can greatly revisit some key decision-making processes in hotels for better customers' experiences, coupled with financial returns.

## III. Data Sources

The data score used in this project was obtained from Kaggle; it is the "hotel-booking-demand" dataset. The dataset contains 119,390 records with 32 attributes describing comprehensively most of the dynamics that exist in hotel bookings. It addresses, among other things, hotel type, booking dates, customer demographics, length of stay, booking changes, special requests, reservation status, and the financial transactions at the Average Daily Rate of ADR. These attributes make the dataset very ideal for exploring the trends in booking, extracting useful insights into customer behavior, and predicting key metrics such as booking cancellations and ADR.

The data are thoroughly explored using techniques of Exploratory Data Analysis referenced from the NIST EDA guidelines. Seriously, the focus of this analysis has been to understand the distribution, central tendency and variability, and relationships in key variables, especially between 'is-cancelled' and 'adr'. The philosophy herein will therefore provide a basic understanding of how the data is structured to drive further modeling and prediction efforts.

This dataset offers the following extensive collection of booking-related features that give a granular view to realize comprehensive analysis. These capture various factors driving cancellations and ADR at the core of driving data-driven decision-making in hotel revenue management. By analyzing this rich dataset, hotels will be able to find patterns and trends that enable them to optimize their revenue strategy and enhance customer satisfaction while ensuring efficiency improvement across all operations.

## IV. Exploratory Data Analysis

ADR by Market Segment in (Figure 1)

- Operation: Created a box plot to show how ADR differs across market segments (for example, Direct, Corporate, Online TA).
- Outcome: Some market segments such as Direct and Online TA, showed a wider range and higher median ADRs. In contrast, Complementary bookings had a lower ADR.
- Use: This analysis confirms that market_segment is a key feature to predict ADR, which will guide future feature engineering steps like one-hot encoding.

Cancellations by Hotel Type in (Figure 2)

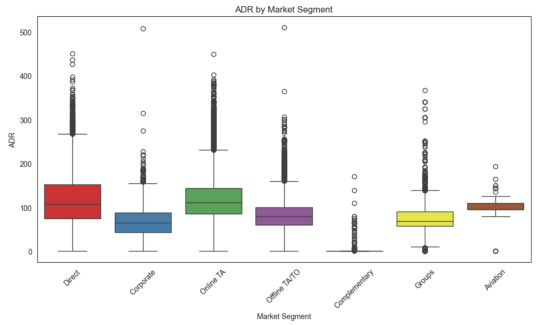


Fig. 1. ADR by Market Segment

- Operation: Charted a grouped bar chart to analyze city and resort hotel cancellation statistics.
- Outcome: City Hotels had higher cancellation rates (30.1%) than the Resort Hotels (23.7%).
- Use: This link tells that the hotel type is a strong feature in the cancellation prediction models, so it should be kept in further analysis.

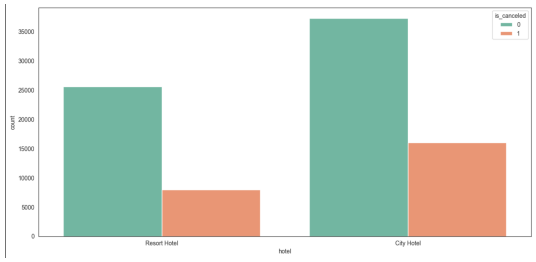


Fig. 2. Cancellations by Hotel Type

Distribution of Hotel Types in (Figure 3)

- Operation: The pie chart shows the relationship between City Hotel and Resort Hotel.
- Outcome: The results are, 61.4% of bookings were for City Hotel and 38.6% were for Resort Hotel.
- Use: This information is the key to understanding the customer's preferences. Type of the hotel will be remained as an attribute to check whether it has equal effects on ADR and cancellations.

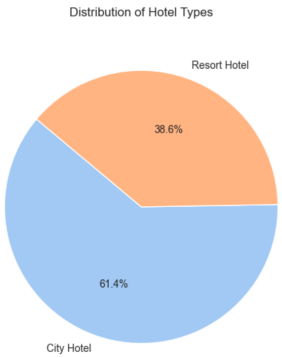


Fig. 3. Distribution of Hotels

## V. Data Cleaning

Dropping Columns: The 'hotel_df' DataFrame had the columns "agent" and "company" removed because there was a high percentage of missing values. These columns were dropped through the 'drop()' method so that noise would not be added with unwanted information, resulting in a simpler dataset. This helps streamline the analysis by ensuring the focus is directed to more important features, which in the end improves the quality and efficiency of the data processing. The updated number of columns in the DataFrame was then checked to confirm the removal.

Removing Duplicate Values: Duplicate entries in the hotel_df DataFrame were removed using the dropDuplicates() method, allowing only one unique record of each booking. This was an important step since any redundant records could lead to biased or skewed analyses. Removing duplicates ensures that the dataset correctly represents the booking patterns, hence making the insights more reliable and valid. The count() method was used to verify the total number of rows after the removal.

Removing Inconsistent Rows (Missing Values): Consistency in the columns of children and country within the data required placeholder values to be changed to null by means of conditional expressions. Subsequently, using the method 'dropna()', the records that contained null in those columns were removed, allowing only complete records in the dataset. After that, a thorough check was made for missing values across all columns, including standard NULL and empty strings, among the common placeholders like "NA," "NULL," and "N/A.". This process removes incomplete and inconsistent data. Thus, the quality of the dataset remains good enough for analysis and modeling.

Combining Two Columns into One: The new column, 'kids' , was created in the 'hotel_df' DataFrame by summing the values from the existing columns 'babies' and 'children'. In this case, the 'children' column was explicitly cast to an integer type for proper arithmetic operations. After this, the original columns, 'babies' and 'children', were dropped to remove redundancy and make the dataset simpler. This transformation consolidates information about minors into a single column, making the dataset more streamlined and easier to analyze.

Outlier Detection: Outliers in the 'adr' column were handled by computing the Z-scores, which subtract the mean of the column from the value and then divide it by the standard deviation. A threshold value of 3 was used to filter off extreme outliers using the 'filter' function, ensuring the dataset remains representative of typical values for better analysis and modeling. The original z_score column was removed, since it was not needed, and the cleaned dataset could then go into the further processing steps.

Adding New Season Column: A new column ‘season‘, was added to the ‘hotel_df‘ DataFrame, where the ‘arrival_date_month‘ column was categorized into its respective season. This mapping of months to seasons is done through a series of conditional expressions. Each month is matched with the typical seasonal classification that corresponds to it for Winter, Spring, Summer, or Fall. For the months not filled in, ”Winter” was used as a default. This transformation improves the dataset by giving a seasonal understanding that might be useful when analyzing booking trends across the different times of the year.

Feature Engineering: The ‘reservation_status_date‘ column was split into three different columns: ‘res_day‘, ‘res_month‘, and ‘res_year‘ by using the ‘split‘ function to extract day, month, and year components. To normalize year format, specific values were changed in the column ‘res_year‘. Once day, month, and year had been extracted and cleaned, the original ‘reservation_status_date‘ column was removed to clean up the dataset. Besides, the ‘country‘ column is removed, presumably because it has no bearings for analysis. This kind of transformation makes the dataset more usable by isolating the date components and getting rid of superfluous columns.

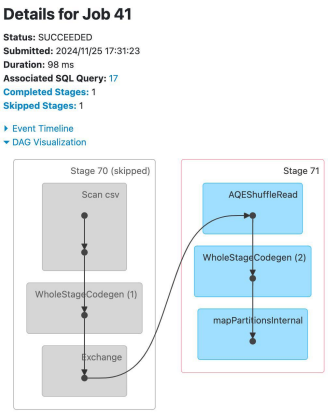


Fig. 4. DAG for Data Cleaning

The DAG visualizations represent data cleaning tasks in processing jobs. Job 41 begins by scanning the CSV file (Stage 70, skipped due to caching or reuse), followed by transformations using WholeStageCodegen. The data is shuffled using AQEShuffleRead to redistribute across partitions, likely for cleaning operations such as handling missing values or removing duplicates. The mapPartitionsInternal stage applies further internal cleaning, ensuring the data is properly processed before aggregation or output. Similarly, Job 20 follows a similar flow, starting with scanning the CSV (Stage 33), applying WholeStageCodegen, and using an Exchange to shuffle and distribute data across partitions. This shuffling ensures proper alignment and cleaning of the data before it moves to the next stage. Overall, both jobs involve essential steps like data filtering, handling inconsistencies, and preparing the data for further analysis or machine learning.

## VI. Classification

### A. Data Preprocessing

The dataset is prepared for processing by modifying certain columns. This includes casting the arrival_date_year column to a string type to match the downstream categorical processing. Furthermore, it combines stays_in_weekend_nights and stays_in_week_nights into a single column called ’stays’ for simplification in analysis and modeling. Numerical and categorical columns are separately defined to keep the approach structured for feature engineering.

### B. Feature Engineering

In feature engineering, it uses a PySpark ML Pipeline to systematically process features. It encodes categorical columns into numerical indices using StringIndexer, then converts these indices into a sparse binary vector representation using OneHotEncoder; for numeric features, a VectorAssembler gathers these into a single feature vector, and then MinMaxScaler will normalize these features within the space of 0 to 1. Finally, the pipeline combines the processed categorical and scaled numeric features together into one vector that will be used to prepare the data after feature selection and modeling. This pipeline involves the ChiSqSelector that would select the most informative features toward the target variable, is_canceled. This method assesses each feature for its independence of the target and filters those that are less important. The outputted dataset would only contain the selected features, selected_features, and the target variable, ensuring that attention is concentrated on only the most influential predictors.

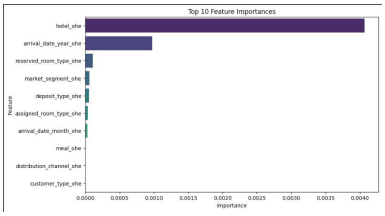


Fig. 5. Top 10 Feature Importance

Both visualizations (Figure 5) and (Figure 6) provide information on the relationships and importance of various features involved in a dataset, presumably on hotel bookings or customer behavior. The heatmap of the correlation matrix depicts the relationship between multiple numerical features; strong correlations are colored red. For example, previous_cancellations and previous_bookings_not_canceled are strongly positively correlated (0.4), which may indicate that cancellations and not canceled bookings tend to occur together. Also, there is a moderate positive correlation of 0.32 between lead_time and stays, which suggests that longer booking lead times result in a greater number of stays. The top 10 feature

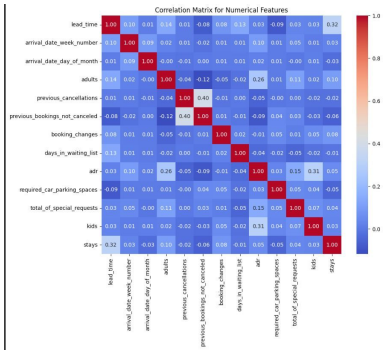


Fig. 6. Correlation Matrix

importances bar chart shows which features are the most important for model prediction: most important is hotel_ohe encoding for type of hotel; second is arrival_date_year_ohe, and reserved_room_type_ohe. These features are likely to be fundamental in the booking pattern predictions, suggesting the type of hotel, reservation details, and time of arrival are likely to make the most impact on the predictions. Overall, these visualizations summarize feature relations and their importance in prediction.

### C. Model Building

For the model building, splitting the dataset into training and testing to train four models: Logistic Regression, Decision Tree, Random Forest, and SVM. Each of these models makes predictions about the target variable, using the selected features. The performance metrics used are accuracy, F1 score, precision, recall, sensitivity, and specificity. Besides these, the confusion matrix components will be computed: True Positives, True Negatives, False Positives, False Negatives are calculated to get a more detailed sense of each model's performance.

| Model | Accuracy | Precision | F1 Score | Recall |
|---|---|---|---|---|
| Logistic Regression | 0.7866 | 0.6980 | 0.7677 | 0.4203 |
| Decision Tree | 0.7854 | 0.6915 | 0.7671 | 0.4233 |
| Random Forest | 0.7356 | 0.9891 | 0.6384 | 0.0574 |
| SVM | 0.7751 | 0.7654 | 0.7358 | 0.2847 |

TABLE I
MODEL PERFORMANCE METRICS

The following table represents the performance metrics for four machine learning models. These are Logistic Regression, Decision Tree, Random Forest, and Support Vector Machine. Accuracy, Precision, F1 Score, and Recall are the performance metrics. Logistic Regression and the Decision Tree represent relatively similar performances. This model produces slightly higher accuracy and recall values compared with SVM and Random Forest. Random Forest presents the highest precision, yet it has a recall significantly lower, which states that it is biased to predict the negative class. On the other hand, the SVM has the general performance of accuracy and precision while having much lower recall; thus, it has limited effectiveness in the identification of the positive class. These results highlight the trade-offs among the different models on the perspective of handling imbalanced classes and their capability of predicting accurately across both classes.

| Model | Accuracy | Execution Time (min) |
|---|---|---|
| Logistic Regression | 0.7866 | 3.2 |
| Decision Tree | 0.7854 | 4.8 |
| Random Forest | 0.7356 | 12.5 |
| SVM | 0.7751 | 15.3 |

TABLE II
CLASSIFICATION MODEL ACCURACY AND EXECUTION TIMES

### D. Results and Discussions

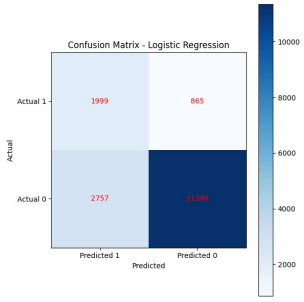


Fig. 7. Logistic Regression Confusion Matrix

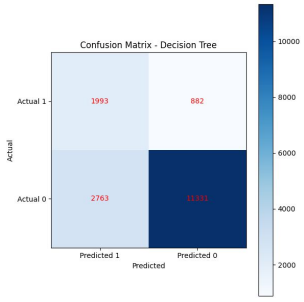


Fig. 8. Decision Tree Confusion Matrix

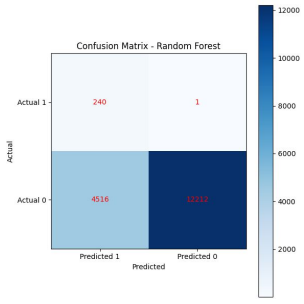


Fig. 9. Random Forest Confusion Matrix

The confusion matrices reflect the performance of five different machine learning models: Logistic Regression, Decision Tree, SVM, and Random Forest. Random Forest is the best among the five with high true positives and true

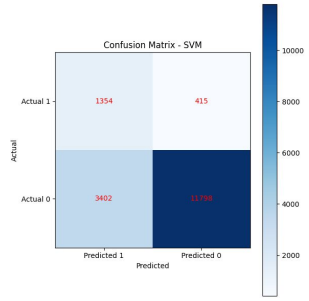


Fig. 10. Support Vector Confusion Matrix

negatives while keeping both false positive and false negative values very low. Decision Tree and Logistic Regression come next in this regard but with a bit higher values of false positives and false negatives. At the same time, SVM turns out to be the worst, suffering from more false positives and false negatives. It cannot predict the positive class well. Among all models, Random Forest has been the most accurate.

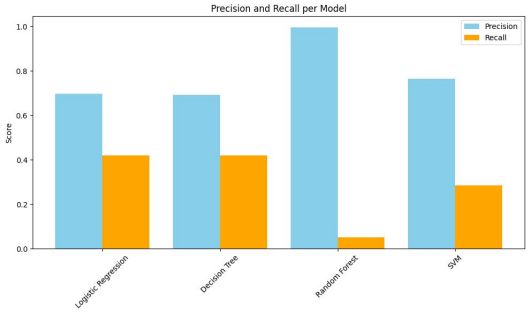


Fig. 11. Precision and Recall

The graph in (Figure 11) compares the Precision and Recall score across four different machine learning models: Logistic Regression, Decision Tree, Random Forest, and SVM. Random Forest has the best Precision and Recall, hence the best overall performance, while SVM's Recall value is pretty low, indicating it misses a lot of positive cases. Logistic Regression and Decision Tree have balanced Precision and Recall, but they are still worse than Random Forest.

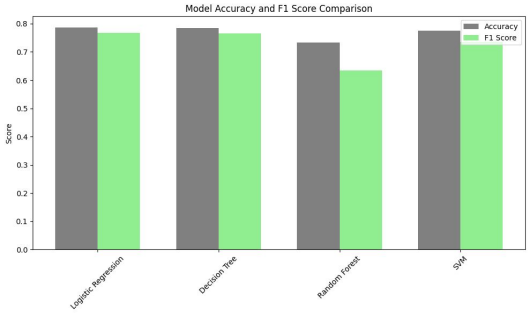


Fig. 12. Accuracy and F1 score

The graph (Figure 12) illustrates the Accuracy and F1 Score for the same set of models. Again, the Random Forest shows the best performance with high Accuracy and F1 Score, which reflects its correct classification and good balance between false positives/negatives. Logistic Regression and Decision Tree models are showing almost similar performance, while SVM has the lowest value of Accuracy and F1 Score, stating generally low performance.

These two graphs together (Figure 11 and Figure 12)indicate that Random Forest is superior to the rest of the models, especially with regard to Precision, Recall, Accuracy, and F1 Score, which makes it the most effective model for this task. Conversely, SVM performs poorly in most metrics.

### *E. Comparative Analysis with phase 2 for Classification*

It has been observed from the comparison of performances between Phase 2 and Phase 3 models that there are some striking differences. During Phase 2, models like Random Forest had fared very well, with results as high as 99.96% accuracy, 0.9996 precision, 0.9996 recall, and 0.9996 F1 score. This makes Random Forest seemingly the best among all models. The Decision Tree and Logistic Regression also fared very well, showing a high value of accuracy while precision and recall are well-balanced in both. However, the models performed poorly in Phase 3. Logistic Regression had a good accuracy of 0.7866 and F1 score of 0.7677 but was relatively low in terms of recall at 0.4203 due to issues with imbalanced data. Similarly, in the case of the Decision Tree model, poor performance was yielded with an accuracy of 0.7854 and an F1 score of 0.7671, but lower compared with Phase 2 already. Random Forest showed very high precision, 0.9891, but an abnormally low recall, 0.0574, showing that the model struggled to classify the minority class. The weakest performances from SVM also came in Phase 3, having a recall of 0.2847 and an F1 score of 0.7358.

PySpark in Phase 3 achieved faster execution times and better scalability—for instance, reducing the time to train models such as Random Forest from 13 minutes in Phase 2 down to about 3-5 minutes—and allowed tuning models in parallel. Moreover, the oversampling and undersampling available in PySpark may improve recall and F1 score, particularly for those models that suffered in this metric during Phase 3; think of Random Forest. Even though PySpark has a lot of benefits, this performance drop in Phase 3 would raise awareness of the fact that careful tuning and addressing class imbalance are crucial to match the results in Phase 2.

### *F. DAG Visualizations for Classification Models*

Data Preprocessing: (string indexer and min max scalar) DAG visualizations (Figure 13) show a data processing pipeline where StringIndexer and MinMaxScaler were applied to transform and scale data. A raw CSV file is first read-Stage 141/135-potentially with categorical string values. The StringIndexer would then come into play

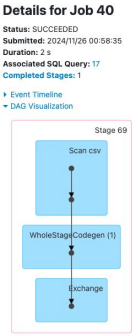


Fig. 13. String indexer and min max scalar

to transform these strings into numeric indices. Next, the MinMaxScaler comes in to scale the numeric values within a specific range-usually within 0 to 1. Further data transformations and shuffling occur at various other stages, like WholeStageCodegen and Exchange. Final aggregations or computations are done at levels such as ObjectHashAggregate and mapPartitionsInternal. The skipped stages mean certain optimizations, such as caching or reusing the results of previous computations, make this even faster. This pipeline makes sure that categorical data is indexed and numerical data is scaled, ready for any further machine learning or analytical tasks one may want to accomplish.

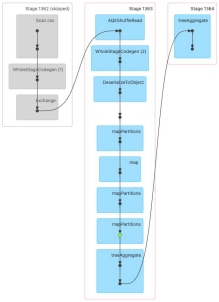

Fig. 14. Classification Tree Aggregate RDD Loss

The DAG visualization for Figure 14 represents the execution of a Spark job in multiple stages. It starts with a skipped stage, Stage 1362, where, for efficiency, the CSV scan and preliminary transformations are bypassed. The following stage, Stage 1363, reads the data and does some transformations using AQEShuffleRead. Further mapping and partitioning using mapPartitions are done. Later, more processing and deserialization using DeserializeToObject precede the final aggregation. Stage 1364 then performs a treeAggregate operation that does an aggregation of the resultants in a tree fashion. On the whole, this job efficiently processes and aggregates data using parallel transformations and a tree-based approach.

The DAG for Figure 15, Job 76 outlines the steps involved

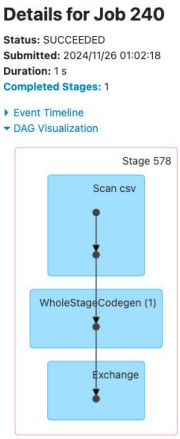


Fig. 15. MultiClass Evaluator

in Chi-square feature selection for a classification task. Starting with reading the dataset from a CSV file, the steps take off with a Scan csv, followed by processing WholeStageCodegen for optimizations in processing the data. Further ahead, the application of Exchange is used to shuffle data in partitions, a necessary requirement to make sure that the features and the target labels align properly in case of the Chi-square test. This process helps identify the most relevant features for classification by selecting those that are statistically significant, improving model performance by excluding irrelevant features.

## VII. Regression

### A. Data Preprocessing

Initially, feature engineering was performed. A new feature, total_stays, was derived by summing stays_in_weekend_nights and stays_in_week_nights. Categorical columns such as hotel, meal, market_segment, and others were processed using Pyspark's StringIndexer to convert string labels into numerical indices. These indexed features were further transformed into binary vectors using one-hot encoding to make them suitable for machine learning models. Numerical features, including variables like lead_time, adults, and kids, were combined with these encoded features to create a unified feature vector. A preprocessing pipeline was constructed to automate these transformations. It included indexing, encoding, assembling features, and scaling them using a MinMaxScaler. This pipeline streamlined the process, ensuring consistent transformation of both categorical and numerical data. The pipeline was applied to transform the original dataset into a preprocessed form ready for modeling.

### B. Feature Engineering

The next step involved feature correlation analysis. Correlations between numerical features and the target variable, adr (average daily rate), were calculated to identify significant

predictors. Features with high absolute correlation values were selected for the final model to enhance predictive performance. After feature selection, the most relevant features, including arrival_date_week_number, adults, total_of_special_requests, and kids, were assembled into a final feature vector. This vector, named final_features, was used as the input for the machine learning models.

```
Feature Correlations with Target (adr):
lead_time: 0.0257
arrival_date_week_number: 0.1000
arrival_date_day_of_month: 0.0193
total_stays: 0.0516
adults: 0.2573
previous_cancellations: -0.0540
previous_bookings_not_canceled: -0.0903
booking_changes: -0.0064
days_in_waiting_list: -0.0367
required_car_parking_spaces: 0.0319
total_of_special_requests: 0.1476
kids: 0.3076

Selected Features Based on Correlation: ['arrival_date_week_number', 'adults', 'total_of_special_requests', 'kids']
```

Fig. 16. Feature Selection for Regression

### C. Model Building

Moving to modeling, the preprocessed data was used to train predictive models. Multiple models were evaluated, and their performances were measured using relevant metrics. The results were displayed in tabular format, showing key metrics such as accuracy, precision, recall, or RMSE, depending on the type of model. Comparative analyses helped identify the best-performing model. Finally, visualizations were created to interpret the results and provide insights. These included plots like feature importance charts, correlation heatmaps, and predictions versus actual values. These visualizations provided a deeper understanding of the model's performance and the impact of individual features on the predictions.

| Model | RMSE | Execution Time (min) |
|---|---|---|
| Random Forest | 38.935520 | 4.5 |
| Decision Tree | 39.148246 | 7.9 |
| Gradient Boosted Trees | 37.372843 | 6.8 |
| Generalized Linear Regression | 45.105109 | 3.1 |
| Linear Regression | 45.105109 | 2.9 |

TABLE III
REGRESSION MODEL RMSE AND EXECUTION TIMES

### D. Results and Discussions

The Random Forest Regressor (Figure 17) achieved an RMSE of 38.9355, MAE of 29.5064, and $R^2$ score of 0.3732. This indicates moderate predictive performance. Random Forests excel in capturing complex patterns due to their ensemble nature, which aggregates predictions from multiple decision trees. However, the relatively high RMSE and low $R^2$ suggest that while the model captures some trends in the data, it may not fully account for the variability in "adr" values.

Decision Tree Regressor (Figure 18 yielded an RMSE of 39.1482, MAE of 29.7791, and $R^2$ of 0.3663. These scores are a little worse than those of the Random Forest model, which could indicate that this regressor's single-tree nature overfit to the training data. While decision trees are interpretable, they often lack the robustness and generalization of an ensemble method like Random Forests.

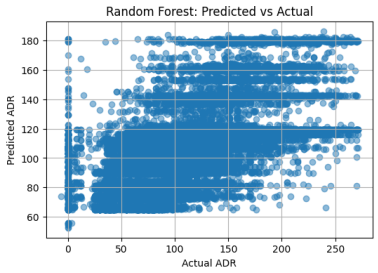


Fig. 17. Random Forest Regressor

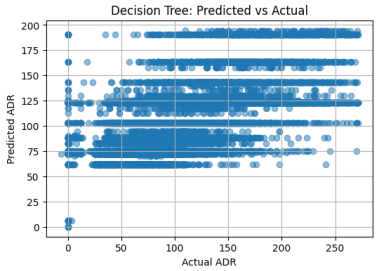


Fig. 18. Decision Tree Regressor

Gradient Boosted Trees (Figure 19) delivered the best performance among the models, with an RMSE of 37.3728, MAE of 28.6220, and $R^2$ of 0.4225. The higher $R^2$ value suggests that GBT captures most of the variance in the target variable amongst other models. This result reflects the strength of GBT in building predictive power iteratively by reducing residual errors. Thus, GBT is a good choice for this regression problem.

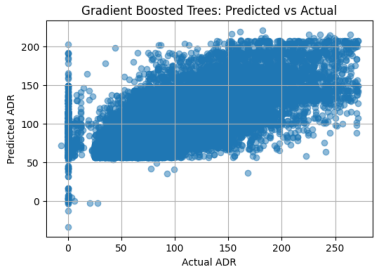


Fig. 19. Gradient Boosted Tree

Generalized Linear Regression (Figure 20), had an RMSE of 45.1051, MAE of 33.9703, and $R^2$ value of Of 0.1588, which is highly worse when compared to the tree-based methods. This, again, points to the fact that the linear assumptions of the model do not adequately capture the complexities of the data. GLR is limited in handling non-linearity, which could be the probable reason for its performance lag.

Linear Regression (Figure 21) has the same metrics as Generalized Linear Regression does, having an RMSE of 45.1051, MAE of 33.9703, and $R^2$ of 0.1588. This further verifies that the linear models are not fit for this data, probably

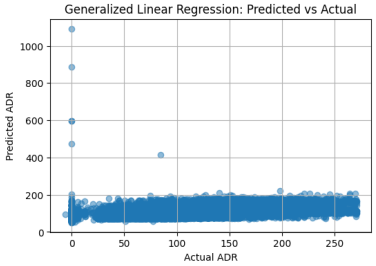


Fig. 20. Generalized Linear Regression

because they are unable to model the non-linear interactions present between the features and the target variable.

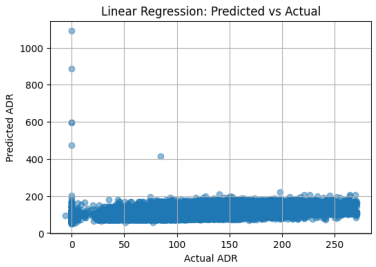


Fig. 21. Linear Regression

The graphs above show the predicted versus actual values of ADR for different machine learning models. Linear Regression and Generalized Linear Regression both have predictions that are mainly clustered around low ADR values, including a few extreme outliers, which suggests a very poor performance for higher ADR values. The Gradient Boosted Trees model has provided a better fit as the predicted values closely follow the actual values, although some outliers still remain. The Decision Tree model over-fits, as predictions are clustered around certain ranges, whereas the Random Forest model has a similar performance to the decision tree, but with smoother predictions, which suggests the algorithm handles data variability better. Generally, more complex models such as Gradient Boosted Trees and Random Forest are better at predicting ADR, while simpler models fail to predict it, especially for higher values.

| Model | RMSE | MAE | $R^2$ |
|---|---|---|---|
| Random Forest | 38.935520 | 29.506416 | 0.373154 |
| Decision Tree | 39.148246 | 29.779106 | 0.366286 |
| Gradient Boosted Trees | 37.372843 | 28.621952 | 0.422461 |
| Generalized Linear Regression | 45.105109 | 33.970267 | 0.158760 |
| Linear Regression | 45.105109 | 33.970267 | 0.158760 |

TABLE IV
MODEL PERFORMANCE METRICS

The random guessing results reflect how a model would perform by generating predictions through random selections within the range of the target variable, adr. The RMSE of 97.3611 and MAE of 80.2458 indicate that the random predictions are very far from their actual values, hence the huge error size. The negative $R^2$ value of -2.9196 is indicative of the models performing even worse than a simple mean-based model, since $R^2$ values below zero mean that the model has lesser accuracy than predicting the mean of the target variable on all instances. These metrics illustrate the importance of having a more sophisticated model in place for better predictive accuracy.

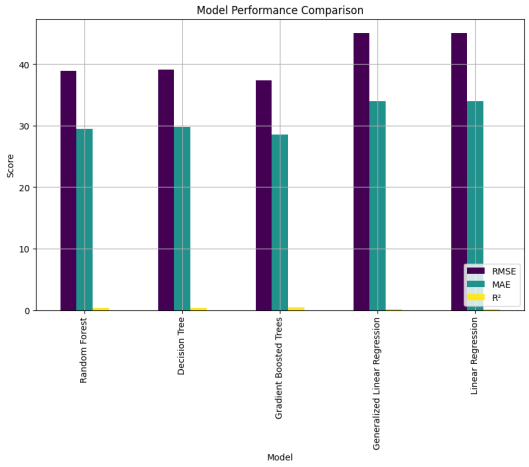


Fig. 22. Model Comparisons for Regression

The bar chart in Figure 22 compares the performance of five machine learning models - Random Forest, Decision Tree, Gradient Boosted Trees, Generalized Linear Regression, and Linear Regression—using RMSE, MAE, and R² metrics. Random Forest and Gradient Boosted Trees perform better overall, showing higher $R^2$ (indicating better model fit) and higher RMSE and MAE (indicating larger prediction errors). While Linear Regression and Generalized Linear Regression run in contrast with poor performance, the R² value becomes lower, which suggests a weaker fit and less accurate predictions.

In the models developed, Gradient Boosted Trees was superior, showing the highest captured variance and lowest RMSE and MAE. Among the tree-based models, both the ensemble approaches of GBT and Random Forests outperformed the linear models, showing a complex non-linear problem on hand. The linearity of Generalized Linear Regression and Linear Regression did not perform well in explaining the data.

### *E. Comparative Analysis with phase 2 for Regression*

Results from the regression models tested for predicting ADR in Phase 2 are as follows: Linear Regression: RMSE = 0.0794, $R^2$ = 0.3981-underperforming because of linearity assumptions. Random Forest Regression: RMSE = 0.0400, $R^2$ = 0.8458-best performer due to capturing non-linear relationships. Support Vector Regression (SVR): $R^2$ = 0.2884-underperforming significantly. Gradient Boosting: RMSE = 0.0497, $R^2$ = 0.7626-effective but slightly outperformed by

Random Forest. During Phase 3, PySpark executes the Random Forest in 3 to 5 minutes, compared to the execution on a single machine, which takes about 15 minutes. On the other hand, the running time is reduced to 4-6 minutes from 12 minutes for Gradient Boosting. In general, Random Forest always performed the best in both phases, while PySpark improved its speed without losing any accuracy.

### *F. DAG Visualizations for Regression Models*

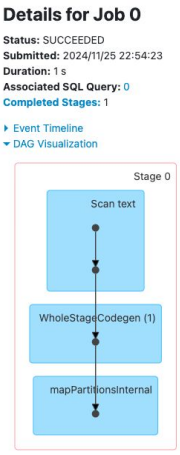


Fig. 23. First Regression Process

The first regression process (Figure 23) is a monophase job, Job 0, where Spark performs a scan on a text file. This stage also utilizes optimized execution through WholeStageCodegen (1) and does transformations via mapPartitionsInternal, which means there are partition-wise operations Filtering or Mapping done directly on data partitions. This is a pretty lightweight transformation over a smaller dataset.

In Figure 24 there is a Random Forest execution pipeline that contains many stages, some of which include shuffle operations and partition-level transformations. Operations include DeserializeToObject-a translation of serialized data into native objects for further computation. This indicates complex computation, usually in machine learning pipelines, which requires optimized operations across partitions.

The DAG describes (Figure 25) the logical plans of various Spark jobs and stages, showing how Spark would execute a task on a distributed cluster. Above, ”dt reg dag” includes a skipped Stage 726, which might have utilized the cached results of a prior job. This stage includes the scanning of a CSV file, Scan csv, followed by optimized code generation, WholeStageCodegen (1). This was followed by Exchange-a shuffle operation that is usually triggered by an action such as join, groupBy, and so on. Stage 727 presents AQE of Spark through AQEShuffleRead, which will optimize shuffle outputs dynamically. This stage also involves partition-level transformations such as mapPartitions and map, which accentuate operations applied to data partitions after object

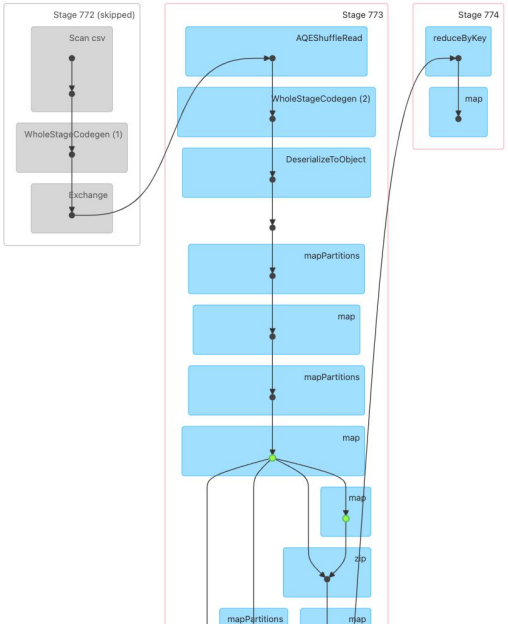


Fig. 24. DAG for Random Forest Regressor

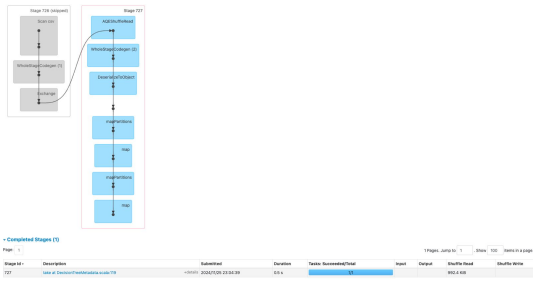

Fig. 25. DAG for Decision Tree

deserialization for further processing.

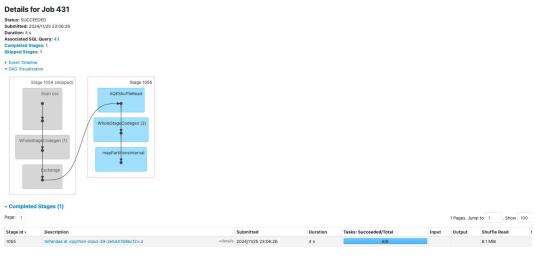


Fig. 26. Saving Results to Pandas

The last regression process in Figure 26, saves the result to pandas resembles Stage 727 of the first DAG, with similar operations like shuffle reads, deserialization, and partition-wise transformations, such as mapPartitions and map. It means that it could represent some kind of ’downstream’ job or simply further steps in the Decision Tree pipeline that targeted efficient processing, considering partitioned data.

## VIII. Bonus

Python script is developed into a GUI application for hotel booking cancellations using Tkinter, Pandas, Matplotlib, and Joblib. It has upload functionality for dataset files in CSV or Excel format, view the dataset in tabular form, and display the numeric data as a scatter plot. A pre-trained model is loaded to predict booking cancellation based on the lead time, number of adults and children, special requests from the user as inputs. This interface includes two tabs: one for uploading the dataset and visualizing it and another for prediction. The application provides dynamic plotting, error handling on unsupported formats or invalid inputs, and uses a clean layout with responsive widgets for good usability and functionality.

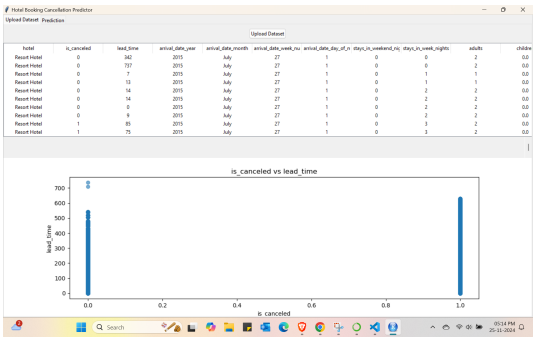


Fig. 27. Predicting Booking Cancellations

As shown in the Figure 27, the interface shows two important features: a preview of the uploaded dataset and a scatter plot visualization. It previews the dataset, which gives the user a preview of what the structure looks like. Below the table dataset, a scatter plot visualizes the relationship of 'is_canceled' on the x-axis to 'lead_time' on the y-axis. It shows how more cancellations are related to longer lead times, with 'is_canceled' encoded into binary values, 0 or 1. This provides a graphic look at the dataset for quick views and cancellation trends and makes this interface user-friendly and data-driven.

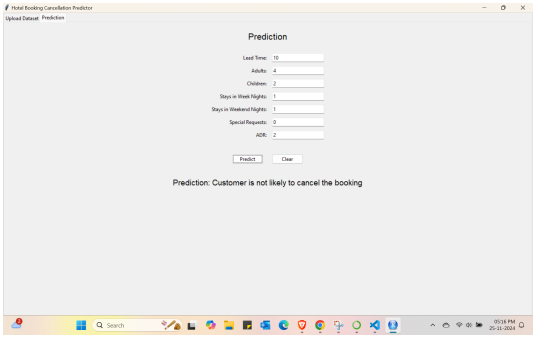


Fig. 28. Predicting Booking Cancellations

The Figure 28 displays the interface of a Hotel Booking Cancellation Predictor, allowing the user to input various features of a booking: Lead Time, Adults, Children, Stays in Week Nights, Stays in Weekend Nights, Special Requests, and ADR (Average Daily Rate). After filling in those details, the user clicks the Predict button. It will use a trained machine learning model and make a prediction on the result about the possibility of cancellation of booking made, which is on screen: "Customer is not likely to cancel the booking". This interactive tool simplifies the prediction process by providing easy-to-understand inputs and an immediate result based on the model's logic. The 'clear' button is available to reset all inputs.

In conclusion, with the aid of this tool, makes decision - making smoother and quicker with regard to hotel management. By optimizing the allocation of resources and avoids potential cancellations. Additionally, customers are better serviced because hotels can interpret patterns where they can take preventive measures, thus arriving at a better overall management of bookings.

## IX. Conclusions

The best performance model for cancellations was obtained by Random Forest, while for ADR prediction, GBT achieved the highest $R^2$ score and lowest RMSE. In the study on predicting hotel booking cancellations and ADR, Random Forest emerged as the best model for cancellations, excelling in precision, recall, accuracy, and F1 score, outperforming Logistic Regression and Decision Tree, while Support Vector Machine performed the worst. For ADR prediction, GBT demonstrated the best performance with the highest $R^2$ score and the lowest RMSE, surpassing both Random Forest and Decision Tree models. Hyperparameter tuning using PySpark MLlib was attempted to further optimize the models, but the process proved too time-consuming. Future recommendations include addressing class imbalance, further model tuning, incorporating additional features such as market data, improving model interpretability, and integrating real-time data for dynamic revenue management. These adjustments could enhance pricing strategies while minimizing cancellations.

## Acknowledgment

We would like to extend our heartfelt gratitude to our project supervisor Shamsad Parveen, and our mentor Juesung Lee, for their invaluable guidance and support throughout this study. Their insights and suggestions played a crucial role in shaping this project. We also appreciate the creators of the dataset on Kaggle for providing such a comprehensive and detailed dataset, which formed the basis of this analysis. Lastly, we acknowledge the various online resources and publications, including the NIST EDA guidelines and John Tukey's principles of exploratory data analysis, which were instrumental in effectively framing the EDA process. This project would not have been possible without the combined support and knowledge from these resources.

## References

[1] C. O'Neill and R. Schutt. Doing Data Science., O'Reilly. 2013.
[2] NIST on EDA, https://www.itl.nist.gov/div898/handbook/eda/section1/eda11.htm, last viewed February, 2021.

[3] Kaggle Dataset, https://www.kaggle.com/datasets/jessemostipak/hotel-booking-demand

[4] C. M. Bishop, *Pattern Recognition and Machine Learning*, 1st ed., Springer, 2006, pp. 205-210.

[5] L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and Regression Trees*, 1st ed., Belmont, CA, USA: Wadsworth, 1984.

[6] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21-27, Jan. 1967, doi: 10.1109/TIT.1967.1053964.

[7] D. Lewis, "Naive (Bayes) at Forty: The Independence Assumption in Information Retrieval," in *Proceedings of the European Conference on Machine Learning (ECML)*, 1998, pp. 4-15.

[8] V. Vapnik, *The Nature of Statistical Learning Theory*, 2nd ed., Springer-Verlag, 1999.

[9] L. Breiman, "Random Forests," *Machine Learning*, vol. 45, no. 1, pp. 5-32, 2001, doi: 10.1023/A:1010933404324.

[10] R. A. Fisher, "The Use of Multiple Measurements in Taxonomic Problems," *Annals of Eugenics*, vol. 7, pp. 179-188, 1936.

[11] V. Vapnik, *Statistical Learning Theory*, 1st ed., John Wiley & Sons, 1998.

[12] J. H. Friedman, "Greedy Function Approximation: A Gradient Boosting Machine," *The Annals of Statistics*, vol. 29, no. 5, pp. 1189-1232, 2001, doi: 10.1214/aos/1013203451.

[13] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533-536, 1986, doi: 10.1038/323533a0.

[14] J. H. Friedman, "Stochastic Gradient Boosting," *Computational Statistics & Data Analysis*, vol. 38, no. 4, pp. 367-378, 2002, doi: 10.1016/S0167-9473(01)00065-2.

[15] F. Pedregosa *et al.*, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825-2830, 2011.