NYC Taxi Fare Prediction: A Machine Learning Approach

Frederick Apina (2019824)
Universidade da Madeira

***Abstract*—This report presents a comprehensive approach to predicting taxi fare amounts in New York City using machine learning techniques. The project begins with problem formulation and progresses through data cleansing, exploratory data analysis, and advanced feature engineering. Multiple models, including k-Nearest Neighbors, Decision Trees, Gradient Boosting, and Neural Networks, were trained and evaluated on both regression and classification tasks. Among the evaluated models, Gradient Boosting Regressor demonstrated the best performance for fare amount prediction, while Bagging and kNN Classifiers showed high effectiveness in categorizing fare ranges. The report also discusses strategies for operationalising the solution in production, covering deployment, monitoring, scalability, and security considerations. The work lays the foundation for intelligent fare prediction systems with real-world applicability in urban transport services.**

***Index Terms*—Data Science, Machine Learning, Regression, Classification, Taxi Fare Prediction, Exploratory Data Analysis, Feature Engineering, Data Cleansing.**

I. Problem Statement (Phase 1)

In metropolitan cities like New York, taxi fares are determined by various dynamic factors such as distance, time, location, and traffic conditions. Accurately predicting taxi fares is valuable for multiple stakeholders: passengers can better anticipate travel costs, while service providers can enhance pricing transparency and operational efficiency.

This project aims to develop a machine learning-based system to predict the fare amount of NYC yellow taxi rides using available trip data. The primary objective is to frame this as a regression problem, where the continuous variable `fare_amount` is predicted from trip-related features. Additionally, we explore a classification reformulation, where fares are categorized into predefined fare bands (e.g., low, moderate, high, premium).

To tackle this problem, the following steps are taken:

1) **Data Acquisition and Understanding:** Load and inspect NYC Yellow Taxi trip data for 2019 from a public dataset.
2) **Data Preprocessing:** Clean the dataset by handling missing values, removing duplicates and outliers, and addressing data leakage.
3) **Feature Engineering:** Derive meaningful features, including temporal, spatial, and engineered attributes to enhance model learning.
4) **Exploratory Data Analysis (EDA):** Analyze distributions, correlations, and relationships to gain insights into key fare determinants.
5) **Model Development:** Train and validate multiple machine learning models for both regression and classification using cross-validation.
6) **Model Evaluation:** Compare models using standard metrics such as R^2, RMSE, MAE for regression and accuracy, precision, recall, and F1-score for classification.
7) **Operationalisation:** Propose a deployment pipeline including real-time model serving, monitoring, and retraining mechanisms.

Success Criteria: The project will be considered successful if:

- The selected model(s) achieve high predictive accuracy, particularly an R^2 score exceeding 0.95 on the test data for regression.
- The classification model achieves an F1-score above 0.85 across fare classes.

II. Data Analysis and Cleansing (Phase 2)

This phase focuses on preparing the data for analysis and modeling. It involves describing the dataset, preprocessing steps including data cleansing and transformation, conducting exploratory data analysis, and performing hypothesis testing.

A. Dataset Description and Source

The dataset used is the ”New York City Taxi Trips 2019” dataset, sourced from Kaggle [1]. For this project, the data for January 2019 was initially used, stored in an SQLite database file named ‘2019-01.sqlite‘ within a ‘tripdata‘ table. The dataset contains various features related to taxi trips, including pickup and dropoff times and locations, trip distance, passenger count, fare details, and payment types. The target variable for prediction is ‘fare_amount‘.

B. Data Preprocessing

Data preprocessing is a critical step to ensure data quality and suitability for machine learning models. The following subsections detail the steps taken.

1) Data Loading and Initial Inspection: The data was loaded using a custom ‘DataLoader‘ class, which reads data from the SQLite database and splits it into features (X) and labels (y). A 90/10 train-test split was performed with a ‘random_state‘ for reproducibility.

```python
class DataLoader:
    def __init__(self, filename, labels_test, test_size=0.1, random_state=None
        , task='regression'):
        self.filename = filename
        # ... (rest of init)
        self._load_data() # Note: method names with _ are fine inside code

    def _load_data(self):
        conn = sqlite3.connect(self.filename)
        query = "SELECT * FROM tripdata"
        df = pd.read_sql_query(query, conn)
        conn.close()
        X = df.drop(columns=self.labels_test)
        y = df[self.labels_test]
        # ... (train_test_split)
        # ... (assign to self.data_train, etc.)
        # Print initial info, describe, nulls
```

```
        df.info()
        print(df.describe(include='all'))
        print(df.isnull().sum())

# Usage:
# data_loader = DataLoader(filename='../data/2019/2019-01.sqlite', labels_test
      ='fare_amount')
```

Listing 1. DataLoader Class Initialization (Conceptual)

Initial data inspection of the January 2019 data (7,667,792 entries) revealed 18 columns. ‘tpep_pickup_datetime‘ and ‘tpep_dropoff_datetime‘ were objects, ‘store_and_fwd_flag‘ was an object, and the rest were float64. A significant number of missing values were observed in ‘congestion_surcharge‘ (4,855,978 missing). Other columns had no missing values initially.

2) Data Type Conversion and Base Feature Engineering: A ‘DataManipulator‘ class, subclassing ‘DataLoader‘, handled initial feature transformations.

- **Datetime Conversion:** ‘tpep_pickup_datetime‘ and ‘tpep_dropoff_datetime‘ were converted to datetime objects. Errors during conversion were coerced to NaT.
- **Categorical Conversion:** ‘store_and_fwd_flag‘ was mapped (’N’: 0, ’Y’: 1). Columns like ‘vendorid‘, ‘ratecodeid‘, ‘pulocationid‘, ‘dolocationid‘, ‘payment_type‘ were converted to integer type.
- **Base Feature Engineering (Trip Duration):** ‘trip_duration_secs‘ was calculated as the difference between dropoff and pickup times in total seconds.

```
class DataManipulator(DataLoader):
    # ... (init)
    def _convert_features(self): # Method name with _
        datetime_cols = ['tpep_pickup_datetime',
                         'tpep_dropoff_datetime']
        for col in datetime_cols:
            self.data_train[col] = pd.to_datetime(self.data_train[col], errors
      ='coerce')
            # ... (similar for test set)

        self.data_train['store_and_fwd_flag'] = self.data_train['
      store_and_fwd_flag'].map({'N': 0, 'Y': 1})
        # ... (similar for test set)
        categorical_cols = ['vendorid', 'ratecodeid', ...] # Ellipsis for
      brevity
        for col in categorical_cols:
            self.data_train[col] = self.data_train[col].astype('int')
            # ... (similar for test set)

    def _engineer_base_features(self): # Method name with _
        self.data_train['trip_duration_secs'] = (self.data_train['
      tpep_dropoff_datetime'] - self.data_train['tpep_pickup_datetime']).dt.
      total_seconds()
        # ... (similar for test set)
```

Listing 2. Data Type Conversion and Base Feature Engineering (Conceptual Snippet)

3) Handling Data Leakage: Features that are direct components or results of the fare calculation, or known only after the fare is determined, were removed to prevent data leakage. These included ‘total_amount‘, ‘tip_amount‘, ‘extra‘, ‘mta_tax‘, and ‘improvement_surcharge‘.

```
    def _handle_data_leakages(self): # Method name with _
        leakage_cols = ['total_amount', 'tip_amount',
                        'extra', 'mta_tax',
                        'improvement_surcharge']
        self.data_train.drop(columns=leakage_cols, inplace=True)
        self.data_test.drop(columns=leakage_cols, inplace=True)
```

Listing 3. Handling Data Leakage (Conceptual Snippet)

4) Data Cleaning: A ‘DataCleaner‘ class was implemented for further cleaning operations on the training data.

- **Remove Duplicates:** Duplicate rows were removed from the training data.
- **Handle Missing Values:**
  - Columns with more than 50% missing values (e.g., ‘congestion_surcharge‘ post-initial load) were dropped.
  - Subsequently, rows with any remaining missing values (e.g., from datetime coercion or other issues) were dropped. This was the chosen strategy (’drop’).
- **Remove Outliers:** Outliers were removed based on defined criteria for several features based on the information provided about the data and for some features like trip_distance Z-score method was used to remove outliers:
  - ‘fare_amount‘: Kept records with ‘fare_amount‘ $>$ \$0 and ‘fare_amount‘ $<=$ \$300.
  - ‘passenger_count‘: Kept records with ‘passenger_count‘ > 0.
  - ‘ratecodeid‘: Kept records with valid codes [1, 2, 3, 4, 5, 6]. Values like 99 were removed.
  - ‘vendorid‘: Kept records with valid IDs [1, 2]. Values like 4 were removed.
  - ‘trip_distance‘: Kept records with ‘trip_distance‘ > 0 miles and ‘trip_distance‘ $<=$60 miles.
  - ‘trip_duration_secs‘: Kept records with ‘trip_duration_secs‘ $>$ 0 seconds and ‘trip_duration_secs‘ $<=$10000 seconds (approx. 2.77 hours).
  - ‘tolls_amount‘: Kept records with ‘tolls_amount‘ $>=$ \$0 and ‘tolls_amount‘ $<=$ \$7.42.

After these cleaning steps, the training data shape was reduced from (6,901,012,13) to (6,636,341,12) before feature engineering. The ‘congestion_surcharge‘ column was dropped, and rows were removed due to missing values and outlier filtering.

C. Exploratory Data Analysis (EDA)

EDA was conducted using the ‘TaxiEDA‘ class to understand data distributions, relationships, and identify patterns or anomalies.

1) Descriptive Statistics and Visualizations:

- **Target Variable Distribution (‘fare_amount‘):** The distribution of ‘fare_amount‘ was visualized using histograms and boxplots.

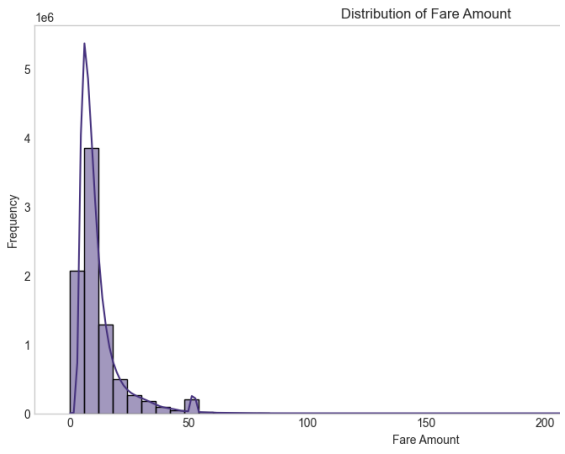

Fig. 1. Distribution of Fare Amount (‘fare_amount‘).

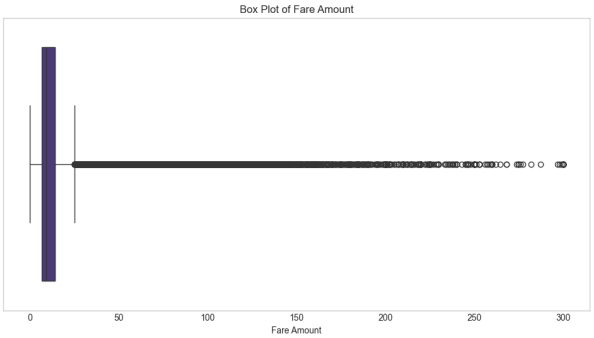

Fig. 2. Box Plot of Fare Amount (‘fare_amount‘).

The *distribution of fare amounts* is highly right-skewed, with a vast majority of fares concentrated at lower values (likely between $0 and $20). There are significant outliers on the higher end, indicating some very expensive fares, as shown by the long tail of the box plot and the spread of the histogram. This suggests that while most transactions are for relatively low amounts, there are occasional instances of much higher fare amounts.

- **Numerical Feature Distributions:** Histograms and boxplots were generated for numerical features like ‘trip_distance‘, ‘passenger_count‘ (after cleaning), and ‘trip_duration_secs‘.

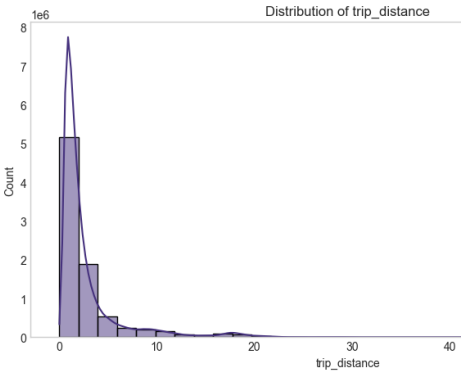

Fig. 3. Distribution of Trip Distance (‘trip_distance‘).

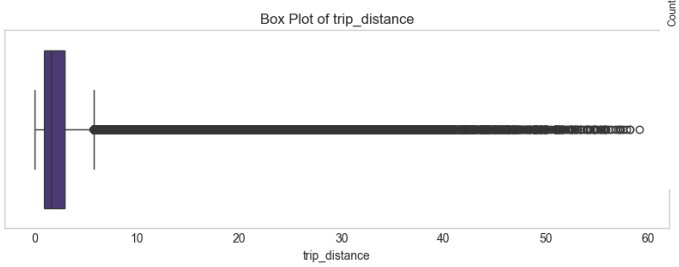

Fig. 4. Box Plot of Trip distance (‘trip_distance‘).

The *trip_distance variable* is heavily concentrated at very low values, with a prominent peak near 0 in the distribution plot, indicating a large number of very short trips. Both plots show a significant right-skew, with a long tail extending to higher distances, suggesting the presence of numerous longer trips, albeit much less frequently than the very short ones. The box plot further emphasizes this by showing a dense interquartile range at the lower end and many outliers extending to much higher distances.

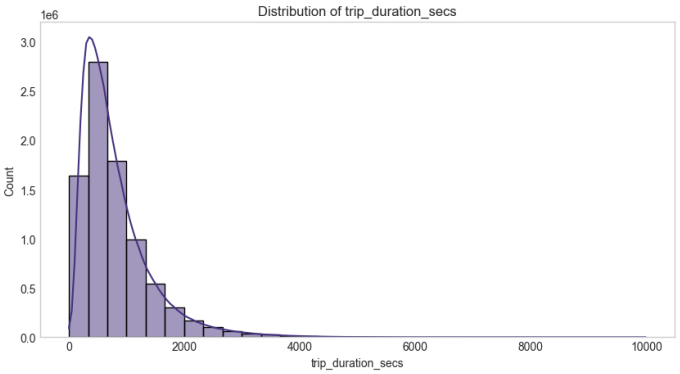

Fig. 5. Distribution of Trip Duration (secs) (‘trip_duration‘).

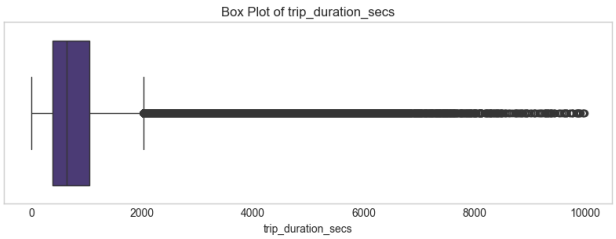

Fig. 6. Box Plot of Trip duration in secs (‘trip_duration‘).

The *trip_duration* variable is highly right-skewed, indicating that most trips are relatively short in duration, with a large concentration of values at the lower end of the spectrum (likely under 1000 seconds). However, there’s a significant number of longer trips, represented by the long tail in both the distribution and box plots, showing that while less frequent, some trips can extend for many thousands of seconds.

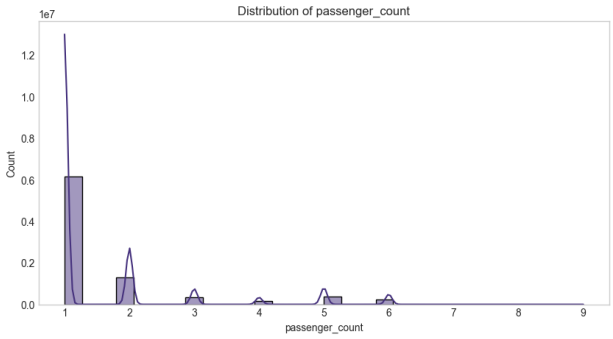

Fig. 7. Distribution of Passenger count

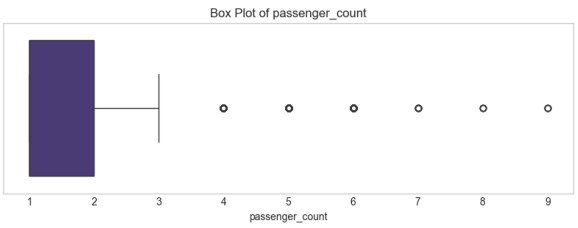

Fig. 8. Box Plot of Passenger count.

The *passenger_count* variable is overwhelmingly dominated by trips with 1 passenger, as evidenced by the very tall bar at '1' in the distribution plot and the compact box plot at the lower end. While there are instances of trips with 2, 3, 4, 5, 6, and even more passengers, these occurrences are significantly less frequent and appear as distinct, smaller peaks in the distribution plot and individual outlier points in the box plot. This indicates that solo travelers are the most common users of this service.

- **Categorical Feature Distributions:** Countplots were used for categorical features like 'vendorid', 'rate-codeid', and 'payment_type'.

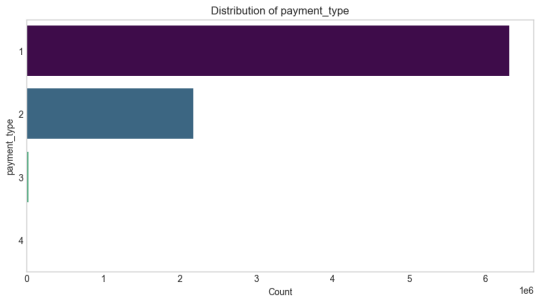

Fig. 9. Payment types Distribution

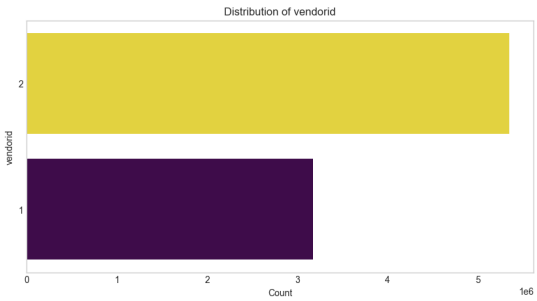

Fig. 10. Vendor types Distribution

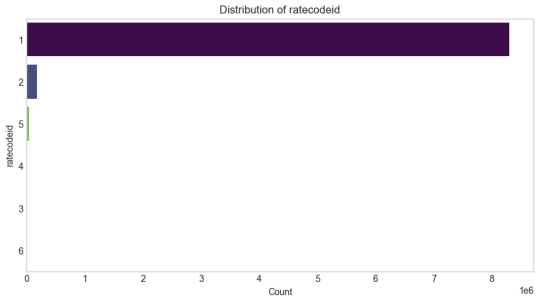

Fig. 11. Rate types Distribution

Interpretation: For payement types there are four types. Most of the trips observed on the plot are dominated by cashless payments (ID 1) followed by cash payments (ID 2) with other forms of payments almost neglible. For vendors, There are two vendors, with Vendor ID 2 (Creative Mobile Technologies, LLC). having significantly more trips than Vendor ID 1 (VeriFone Inc). This suggests that Vendor 2 is either a larger service provider or more frequently used. The overwhelming majority of trips fall under RateCodeID 1. RateCodeID 2 and 5 account for a very small fraction of trips, and RateCodeID 3, 4, and 6 are almost non-existent. This indicates that most trips are charged under the standard rate.

- **Relationships with Target Variable:**
  - *Numerical vs. Target:* Scatter plots (e.g., 'fare_amount' vs. 'trip_distance', 'fare_amount' vs. 'trip_duration_secs') were generated using a sample of the data for performance.

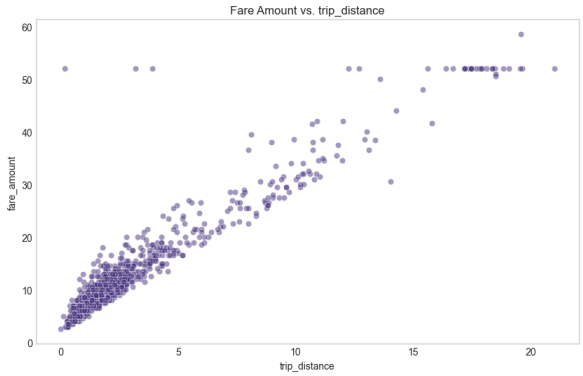

Fig. 12. Scatter plot of Fare Amount vs. Trip Distance

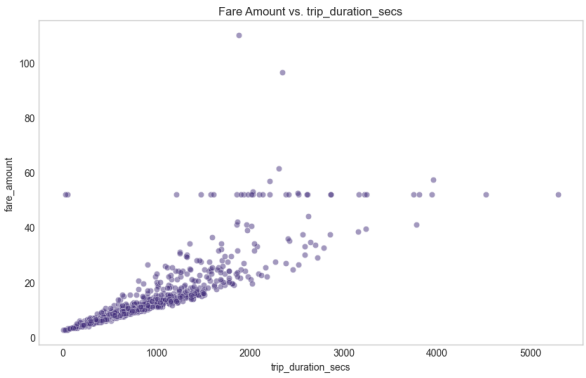

Fig. 13. Scatter plot of Fare Amount vs. Trip Duration

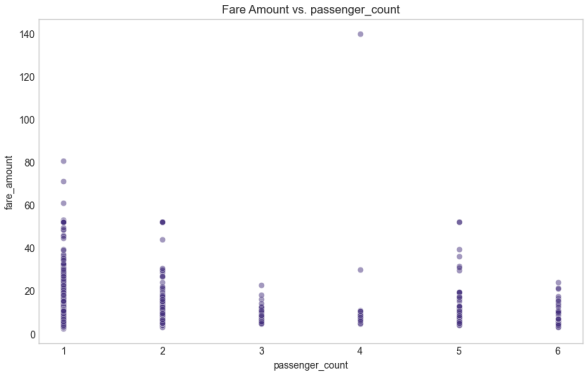

Fig. 14. Scatter plot of Fare Amount vs. Passenger count

For Fare Amount vs. Trip Distance (Fig. 12): There is a clear positive correlation between fare amount and trip distance. As the trip distance

increases, the fare amount tends to increase as well.

Fare Amount vs. Trip Duration (Fig. 13): Similar to trip distance, there is a positive correlation between fare amount and trip duration. Longer trips (in seconds) generally result in higher fares. There are some horizontal lines of points, particularly around the $50-55$ fare mark, suggesting a possible flat rate for certain durations or other factors influencing the fare.

Fare Amount vs. Passenger Count (Fig. 14): There doesn't appear to be a strong, clear correlation between fare amount and passenger count. Fares vary widely across all passenger counts.

– *Categorical vs. Target:* To observe the relationship between categorical variables and the target Boxplots (e.g., 'fare_amount' vs. 'payment_type', 'fare_amount' vs. 'ratecodeid') were generated.

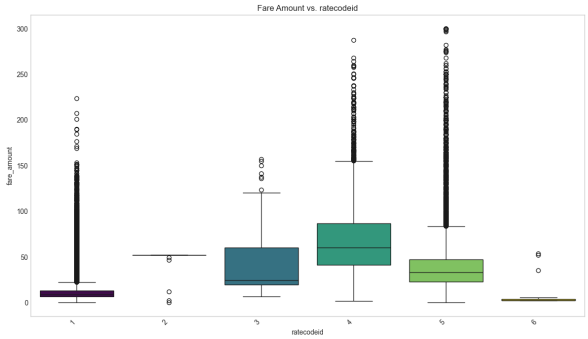

Fig. 15. 'Fare amount' vs. 'RateCodeID'.

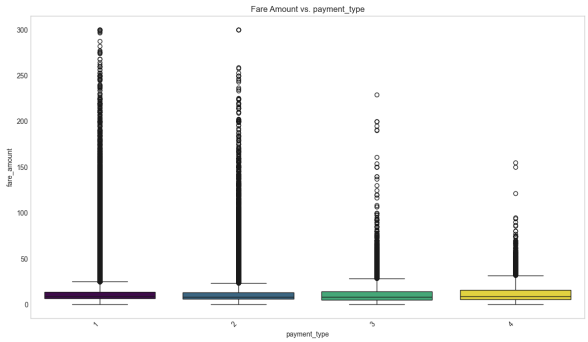

Fig. 16. 'Fare amount vs. Payment type.

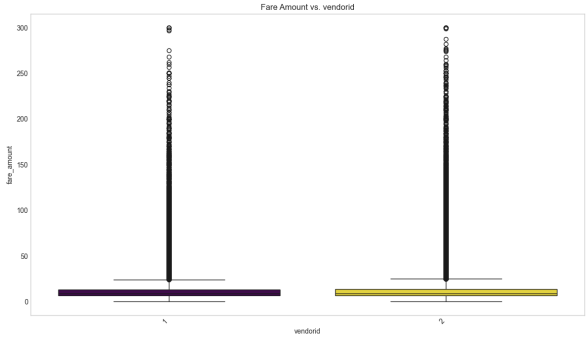

Fig. 17. 'Fare amount vs. Vendor type.

Fare Amount vs. Rate Code ID (Fig. 15): Rate codes 1 and 6 have the lowest median fares and narrowest distributions. Rate codes 3, 4, and 5 show progressively higher median fares and wider distributions. Rate code 4 has the highest median fare. Rate code 2 has a slightly higher median than 1 but a very tight distribution, with few outliers shown. Many rate codes (1, 3, 4, 5) have numerous high-fare outliers.

Fare Amount vs. Payment Type (Fig. 16): Payment types 1 (cashless) and 2 (cash) have very similar median fares and distributions, with payment type 1 showing slightly more high-fare outliers. Payment types 3 (no charge) and 4 (dispute) also have similar median fares to each other, which are slightly lower than types 1 and 2. They also show fewer extreme outliers compared to types 1 and 2.

Fare Amount vs. Vendor ID: Both vendors show very similar fare distributions, with medians and interquartile ranges appearing nearly identical. Both have many high-fare outliers.

- **Correlation Analysis:** A correlation heatmap of numerical features (including the target variable 'fare_amount' after cleaning) was plotted. And the observations are described below;

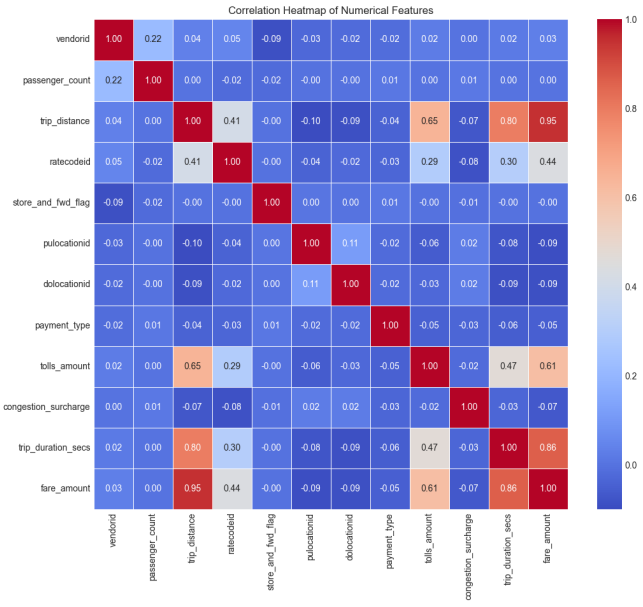

Fig. 18. Correlation Heatmap.

Strong Positive Correlation: fare_amount is strongly positively correlated with trip_distance (0.95) and trip_duration_secs (0.86). This means as distance or duration increases, the fare amount significantly increases.

Moderate Positive Correlation: fare_amount has a moderate positive correlation with tolls_amount (0.61). Higher tolls are associated with higher fares.

Weak Correlation: fare_amount shows weak correlations with ratecodeid (0.44), passenger_count (0.22), and very weak or negligible correlations with other variables like vendorid, store_and_fwd_flag, pulo-

cationid, dolocationid, payment_type, and congestion_surcharge.

2) Dimensionality Reduction: The project plan requires dimensionality reduction using at least one linear (e.g., PCA) and one non-linear method (e.g., UMAP) to identify patterns.

- **Principal Component Analysis (PCA):** To understand the underlying structure and reduce the dimensionality of our numerical features, we applied Principal Component Analysis (PCA). The numerical features considered for PCA were 'trip_distance', 'trip_distance_secs', 'tolls_amount'. The PCA was performed by first fitting a PCA model to determine the number of components needed to explain at least 80% of the variance in the data. This transforms both the training and testing datasets into this reduced dimensional space.

Fig. 19. PCA: Principal Component 1 vs. Principal Component 2.

Insights from PCA: The scatter plot of the first two principal components (PC1 and PC2) in (Fig. 19) provides a visual representation of the data's variance. In summary, while ratecodeid 1.0 and 2.0 are intermingled, the first two principal components effectively highlight differences and create some separation for the other ratecodeid categories, often arranging them into distinct linear patterns or clusters.

D. Hypothesis Testing

The project requires formulating and testing hypotheses. The notebook outlines several hypotheses.

1) Hypotheses Formulation and Test Selection:

1) **H1: trip_distance vs fare_amount.**
   - H_0: There is no significant correlation between trip duration and fare amount.
   - H_A: There is a significant correlation between trip duration and fare amount.
   - *Test Chosen:* Pearson correlation coefficient.
   - *Results*: Correlation coefficient (r) was 0.9488 which is a very strong positive correlation, close to +1. The P-value was 0 which essentially indicating extremely strong evidence against the null hypothesis.
   - *Interpretation*: Since the p-value is below any conventional threshold (0.05), we reject H_0. There is a statistically significant and strong positive linear relationship between trip_distance and fare_amount. This means longer trips tend to cost more, as expected.
2) **H2: fare_amount by ratecodeid.**
   - H_0: The mean fare_amount is the same across all ratecodeid groups.
   - H_A: The mean fare_amount differs for at least one ratecodeid group.
   - *Test Chosen:* ANOVA.
   - *Results*: F-statistic is 997938.8610 which is extremely large, indicating strong group separation. P-value is 0 which this indicates overwhelming evidence against the null hypothesis.
   - *Interpretation*: Since the p-value is effectively 0, we reject H_0. This means there is a statistically significant difference in average fare_amount between at least two ratecodeid groups. In other words, fare prices vary depending on the rate code category, which could reflect different pricing rules.

III. Model Selection Strategy (Phase 3)

Phase 3 transitions from data understanding to preparing for predictive modeling. This involved creating impactful features, identifying suitable machine learning algorithms for both regression and classification tasks, and establishing a robust validation methodology.

A. Feature Engineering

To enhance model performance, over ten new features were systematically engineered. These features aim to capture temporal dynamics, trip-specific details, and cyclical patterns. Well-engineered features are critical for enabling models to learn complex relationships and improve predictive accuracy.

Key engineered features include:

- **Datetime Features:** 'pickup_hour', 'pickup_day', 'pickup_month', 'pickup_is_weekend', 'is_rush_hour' and 'time_of_day_segment' (e.g., 'Morning', 'Evening'). This helps Capture time-dependent variations in demand and traffic.
- **Trip-Specific Features:** 'is_airport_trip' (derived from 'ratecodeid' 2 for JFK and 3 for Newark), 'is_same_location_trip', and 'average_speed_mph'. This account for special fare conditions (airports), potential data anomalies or unique trip types (same location), and traffic congestion (speed).
- **Cyclical Time Features:** 'pickup_hour_sin', 'pickup_hour_cos', 'pickup_day_sin', and 'pickup_day_cos'. These allow models to correctly interpret the cyclical nature of time (e.g., hour 23 being close to hour 0), which is crucial for algorithms sensitive to feature magnitude or ordinality.

```
class TaxiFeatureEngineer:
    def create_datetime_features(self, df):
        df['pickup_hour'] = df['tpep_pickup_datetime'].dt.hour
        df['is_rush_hour'] = ... # Logic for rush hour

```

```
    def create_trip_features(self, df):
        df['is_airport_trip'] = df['ratecodeid'].isin
    ([2, 3]).astype(int)
        df['average_speed_mph'] = ... # Calculated from
     distance and duration

    def create_cyclical_time_features(self, df):
        df['pickup_hour_sin'] = np.sin(2 * np.pi * df['
    pickup_hour'] / 24.0)        # ... and other
    cyclical transformations
```

Listing 4. Feature engineering code snippet

These engineered features provide richer, more informative inputs for the subsequent modeling phase.

B. Examination of Suitable Models

The primary goal is to predict taxi fares, approached as both a regression task (for 'fare_amount') and a classification task (for fare ranges). The selection of models, guided by the project requirements, encompasses a diverse set of algorithms to ensure a comprehensive evaluation. Exploring various model types allows for a robust comparison to identify those best suited for capturing the underlying patterns in the NYC taxi dataset.

The models considered are grouped as follows:

- **k-Nearest Neighbors (kNN):**
  - *Suitability:* Applicable to both regression and classification. It's a non-parametric, instance-based learner that makes predictions based on the 'k' most similar training examples.
  - A kNN algorithm was implemented from scratch in Phase 4, demonstrating a fundamental understanding of its mechanics.
  - *Justification:* Useful for capturing local structures in the data without making strong assumptions about the data distribution.
- **Supervised Learning Models:**
  - *Decision Trees (DecisionTreeRegressor, DecisionTreeClassifier):*
    - *Suitability:* For both regression and classification.
    - *Justification:* They are interpretable, can capture non-linear relationships, and handle numerical and categorical data.
  - *Multi-Layer Perceptron (MLPRegressor, MLPClassifier):*
    - *Suitability:* For both regression and classification. MLP is a type of feedforward artificial neural network.
    - *Justification:* Capable of learning complex, non-linear patterns and can serve as a foundational deep learning approach.
- **Ensemble Models:**
  - *Gradient Boosting (GradientBoostingRegressor, GradientBoostingClassifier):*
    - *Suitability:* For both regression and classification.
    - *Justification:* Known for high performance, it builds models sequentially, with each new model correcting errors of its predecessors. Effective in handling complex datasets and often yields state-of-the-art results.
  - *Bagging (BaggingRegressor, BaggingClassifier):*
    - *Suitability:* For both regression and classification.
    - *Justification:* Bagging methods improve stability and accuracy by training multiple base estimators on different random subsets of data and averaging their predictions, which helps reduce variance.
- **Deep Learning Model:**
  - *Custom Neural Network.*
    - *Suitability:* Primarily for complex regression or classification tasks with large datasets.
    - *Justification:* Deep learning models, such as custom-built neural networks, can automatically learn hierarchical feature representations and model highly intricate non-linear relationships, potentially offering performance gains on complex problems.
- **Clustering Models (KMeans, DBSCAN):**
  - *Suitability:* KMeans and DBSCAN are unsupervised clustering algorithms.
  - *Role in this Project:* While not direct predictive models for supervised regression or classification of fares, they are valuable for *exploratory data analysis* to discover clusters of trips.
  - *Justification:* Understanding the inherent structure of the data through clustering can provide insights that inform feature creation or help in identifying distinct data regimes that might benefit from specialized modeling approaches.

This diverse selection ensures that various algorithmic approaches—ranging from instance-based and linear models to complex tree ensembles and neural networks—are considered, providing a solid basis for identifying the most effective techniques for this prediction problem.

C. Assessment of Model Validation Method

A robust validation strategy is critical for estimating model performance on unseen data and for reliable hyperparameter tuning.

- **Train-Test Split:** An initial split of the data (e.g., 90% training, 10% testing as established in Phase 2) is maintained. *Justification:* The test set is reserved for a final, unbiased evaluation of the chosen model(s) after all training and tuning are complete.
- **k-Fold Cross-Validation:** For model training and hyperparameter tuning, 5-fold cross-validation (as employed with 'GridSearchCV' in Phase 4) is the chosen method. *Justification:* This technique provides a more reliable estimate of model performance by training and evaluating the model on multiple, different subsets of the training data. It mitigates the risk of performance estimates being overly optimistic or pessimistic due to a specific random split for a single validation set. Furthermore, it is integral to systematic hyperparameter optimization processes like 'GridSearchCV', ensuring that selected hyperparameters generalize well across different data samples.

Standard evaluation metrics relevant to each task (e.g., RMSE, R^2 for regression; Accuracy, F1-score for clas-

sification) will be used in conjunction with this validation framework. This approach ensures that model performance is assessed thoroughly and that the selected models are generalizable.

IV. Model Building (Phase 4)

This phase focuses on the implementation, training, and initial evaluation of various machine learning models.

A. kNN Algorithm

The k-Nearest Neighbors (kNN) algorithm was implemented from scratch.

- **Implementation**: A `CustomKNN` class was developed using NumPy for core computations and `sklearn.neighbors.BallTree` for efficient nearest neighbor searches. This custom class inherits from a `BaseModel` class, which handles common functionalities like evaluation and saving/loading models.
- **Data Structure**: The `BallTree` data structure was utilized to optimize the process of finding the k-nearest neighbors, especially for larger datasets.
- **Explanation**:
  - The `CustomKNN` class initializes with a model name, task type (regression or classification), and a list of k-values to test (`k_options`).
  - The `_fit_predict_single_knn` method handles the prediction for a given k, using `BallTree` to find neighbors. For regression, it predicts the mean of neighbor labels; for classification, it uses the mode.
  - The `fit` method performs hyperparameter tuning for 'k' using K-Fold Cross-Validation (manual CV with `N_CV_SPLITS` folds). It iterates through `k_options`, calculates performance metrics (R^2 for regression, Accuracy for classification) for each fold, and selects the 'k' that yields the best average score.
  - The best 'k' and the training data are stored to be used by the `predict` method on new data, which utilizes the pre-fitted `_ball_tree_final`.
- **Results & Discussion** (Table. III) and (Table. IV): In both scenarios of classification and regression, the optimal number of neighbors ('k') was consistently determined to be 7 across all five folds. The KNN Regressor demonstrated strong performance with R2 scores consistently above 0.96 and low error metrics, while the KNN Classifier also showed high efficacy with accuracy scores around 0.98 and robust F1 scores generally exceeding 0.91.

B. Supervised Learning (Other than kNN)

Two other supervised learning models were tested using the scikit-learn library. All sklearn models were wrapped in an `SklearnModel` class, which also inherits from `BaseModel` and incorporates `GridSearchCV` for hyperparameter tuning.

- **Models Tested**:
  1) **Decision Tree**:
     - *Regression (`DecisionTreeRegressor`)*: Used to predict scaled fare amounts.
     - *Classification (`DecisionTreeClassifier`)*: Used to predict fare classes.
  2) **Multi-layer Perceptron (MLP)**:
     - *Regression (`MLPRegressor`)*: Used to predict scaled fare amounts.
     - *Classification (`MLPClassifier`)*: Used to predict fare classes.
- **Hyperparameter Search (`GridSearchCV`)**: For each scikit-learn model, `GridSearchCV` was used with `N_CV_SPLITS` folds to find the best combination of hyperparameters from a predefined `param_grid`.
  - *DecisionTreeRegressor*: Tuned `max_depth`, `min_samples_split`, `min_samples_leaf`, `criterion`, and `max_features`.
  - *MLPRegressor*: Tuned `hidden_layer_sizes`, `learning_rate_init`, `activation`, and `alpha`, with early stopping enabled.
  - *DecisionTreeClassifier*: Tuned `max_depth`, `min_samples_split`, and `min_samples_leaf`.
  - *MLPClassifier*: Tuned `hidden_layer_sizes`, `learning_rate_init`, `activation`, and `alpha`, with early stopping enabled.
- **Discussion of Results** (Table. VI) and (Table. V): The MLP models demonstrated slightly better performance for both regression and classification tasks compared to the Decision Tree models based on the provided fold evaluations.

C. Ensemble Models

Two ensemble models, one using bagging and one using boosting, were chosen and applied for both regression and classification tasks, also utilizing the `SklearnModel` class with `GridSearchCV`.

- **Bagging Model**:
  - *Regression (`BaggingRegressor`)*: Based on decision trees. Hyperparameters tuned included `n_estimators`, `max_samples`, `max_features`, `bootstrap`, and `bootstrap_features`.
  - *Classification (`BaggingClassifier`)*: Based on decision trees. Hyperparameters tuned included `n_estimators`, `max_samples`, `bootstrap`, and `bootstrap_features`.
- **Boosting Model**:
  - *Regression (`GradientBoostingRegressor`)*: Hyperparameters tuned included `n_estimators`, `learning_rate`, `max_depth`, `subsample`, and `max_features`.
  - *Classification (`GradientBoostingClassifier`)*: Hyper-

parameters tuned included `n_estimators`, `learning_rate`, and `max_depth`.

- **Discussion & Results** (Table. X, (Table. IX) and (Table. XIV),(Table. XIII)): The GradientBoostingRegressor demonstrated strong performance, achieving a high R2 score of 0.9843 with parameters 'learning_rate': 0.1, 'max_depth': 3, 'max_features': 0.5, 'n_estimators': 100, 'subsample': 0.8. In classification, the GradientBoostingClassifier reached an F1 score of 0.9443 using 'learning_rate': 0.05, 'max_depth': 5, 'n_estimators': 100. Comparatively, the BaggingRegressor showed slightly lower regression performance with a top R2 score of 0.9379, while the BaggingClassifier achieved its best F1 score of 0.9501 with {'bootstrap': True, 'bootstrap_features': False, 'max_samples': 0.5, 'n_estimators': 50}.

D. Deep Learning Model

A deep learning model was implemented from scratch using TensorFlow/Keras for both regression and classification. This was managed by the `TFDeepLearningModel` class, inheriting from `BaseModel`.

- **Architecture**: A sequential neural network was chosen. The `_build_model_dynamic` method constructs the model with:
  - An input layer with dimension based on the training data.
  - Two dense hidden layers with ReLU activation and L2 kernel regularization, followed by Dropout layers and Batch Normalization.
  - The number of units in these layers was a hyperparameter.
  - An output layer: a single unit with no activation for regression, and a single unit with sigmoid activation.
  - Adam optimizer was used. Loss functions were 'mse' for regression, 'binary_crossentropy' or 'sparse_categorical_crossentropy' for classification.
- **Implementation**: TensorFlow (via Keras API) was used. Callbacks like `EarlyStopping` and `ReduceLROnPlateau` were used during training.
- **Hyperparameter Search**: A manual K-Fold Cross-Validation approach (`N_CV_SPLITS` folds) was used within the `fit` method to tune hyperparameters defined in `tune_params`. For regression and classification, this included `units_layer1`, `batch_size`, and `learning_rate`. The best parameter combination was selected based on average validation performance (R^2 for regression, Accuracy for classification).
- **Discussion of Results**: The DeepLearningRegressor achieved its best R2 score of 0.9207 with parameters 'units_layer1': 64, 'batch_size': 32, 'learning_rate': 0.0001, though performance varied across folds with R2 scores ranging down to 0.8690. For classification, the DeepLearningClassifier showed a top F1 score of 0.7242 using 'units_layer1': 64, 'batch_size': 64, 'learning_rate': 0.01. Overall, the deep learning models demonstrate reasonable regression capabilities but more modest classification performance based on the F1 scores in these evaluations.

E. Clustering

Two clustering algorithms were applied to a subset of the training data (`X_cluster_data`) to identify potential patterns.

- **Algorithms Applied**:
  1) **KMeans (`sklearn.cluster.KMeans`)**:
     - Varied the number of clusters (k) in the range [2, 3, 4, 5].
     - Evaluated using Silhouette Score, Davies-Bouldin Score, and Inertia.
     - The best 'k' was determined by the highest Silhouette Score.
  2) **DBSCAN (`sklearn.cluster.DBSCAN`)**:
     - Varied the `eps` parameter with values [0.5, 1.0], keeping `min_samples` at 5.
     - Evaluated using Silhouette Score and Davies-Bouldin Score (excluding noise points).
- **Pattern Assessment & Discussion**:
  - PCA (2 components) was used to visualize the clusters for both KMeans (best k) and DBSCAN (best eps).
  - Silhouette plots were generated for the best KMeans model to assess cluster cohesion and separation.
  - Cluster characteristics for the best KMeans model were analyzed by examining the deviation of cluster centers from the overall mean for each feature and by analyzing fare distributions within clusters.
  - The K-Means analysis indicated that two clusters (k=2) is optimal, as suggested by the highest Silhouette score and a favorable Davies-Bouldin score at this number. Visualizations using PCA show distinct cluster formations for both K-Means (k=2) and DBSCAN, with the silhouette plot for K-Means confirming good cluster separation as most points in cluster 0 have high silhouette coefficients. Cluster characteristics for the best K-Means model were further explored by analyzing feature deviations and fare distributions. The fare distribution analysis clearly shows that Cluster 1 is associated with significantly higher average fares compared to Cluster 0.

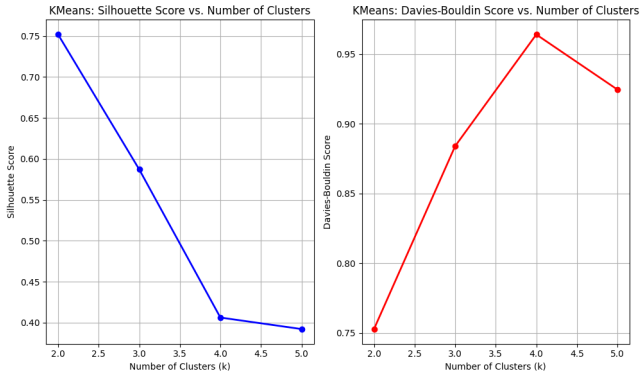

Fig. 20. KMeans: Silhoutte and Davies-Bouldin scores

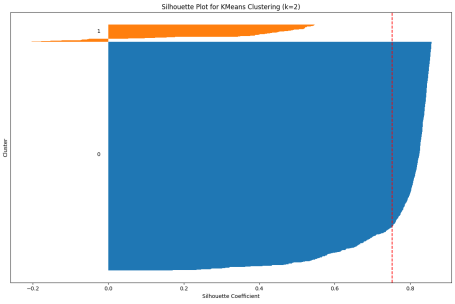

Fig. 21. KMeans (K=2): Silhoutte score plot

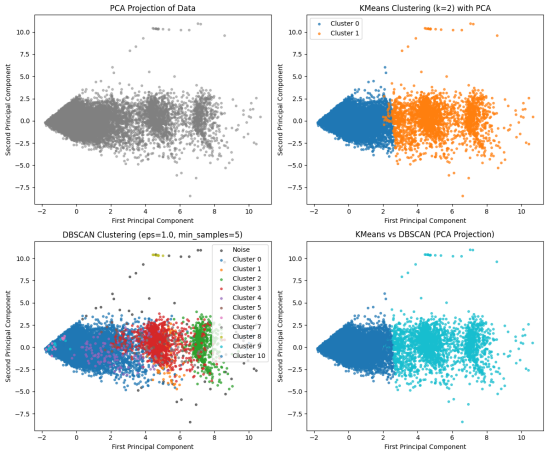

Fig. 22. PCA Projections of KMeans and Silhoutte

V. Model Comparison and Evaluation

This phase consolidates the results from the model building efforts in Phase 4, providing a comparative analysis of the different machine learning models developed for predicting NYC taxi fares. The comparison is segmented into regression and classification tasks, followed by an overall discussion and selection of the most promising models based on the evaluation metrics.

A. Regression Model Comparison

Multiple regression models were implemented and evaluated, including k-Nearest Neighbors (kNN) Regressor, Decision Tree Regressor, Multi-Layer Perceptron (MLP) Regressor, Bagging Regressor, Gradient Boosting Regressor, and a custom Deep Learning Regressor. The primary metrics used for comparison are the R-squared (R^2) score, Root Mean Squared Error (RMSE), and Mean Absolute Error (MAE). Below were the observations;

TABLE I
Regression: Model Comparisons on Test Data

Model Name	R^2	MAE	MSE	RMSE
KNN	0.9340	0.4904	2.8001	7.8407
DecisionTreeRegressor	0.9433	0.8939	2.5966	6.7423
BaggingRegressor	0.9261	1.5673	8.7837	2.9637
GradientBoostingRegressor	0.9597	0.5653	4.7855	2.1876
DeepLearningRegressor	-4.5842	3.0782	689.9637	26.2672
MLPRegressor	-0.8261	2.5673	123.883	11.53

- **kNN Regressor**: Demonstrated strong performance with R^2 scores consistently above 0.93 and low error metrics. The optimal number of neighbors ('k') was found to be 7.
- **Decision Tree Regressor**: This model had the best second R^2 performance metrics.
- **MLP Regressor**: On test data MLP shows a very low $R^2 \approx -0.8$. This might suggest that the model was overfitting.
- **Bagging Regressor**: Showed good performance with a top R^2 score of 0.926.
- **Gradient Boosting Regressor**: Demonstrated stronged performance, achieving a high R^2 score of 0.9597.
- **Deep Learning Regressor**: Achieved the worst R^2 score of -4.5842, suggesting it was not learning well.

B. Classification Model Comparison

For the classification task (predicting fare ranges), models included kNN Classifier, Decision Tree Classifier, MLP Classifier, Bagging Classifier, Gradient Boosting Classifier, and a custom Deep Learning Classifier. The primary metrics discussed in the report are Accuracy and F1-score. Below were the observations;

TABLE II
Classification: Model Comparisons on Test Data

Model Name	Accuracy	Precision	Recall	F1 Score
KNN	0.9774	0.9343	0.8258	0.8588
DecisionTreeClassifier	0.9786	0.9350	0.8259	0.8593
BaggingClassifier	0.9784	0.9096	0.8350	0.8606
GradientBoostingClassifier	0.9780	0.8301	0.8399	0.8349
DeepLearningClassifier	0.9738	0.7591	0.7615	0.7600
MLPClassifier	0.9785	0.9000	0.8298	0.8538

For the classification task, the **Bagging Classifier** (Accuracy ≈ 0.98, F1 > 0.86) and the **KNN** (F1 = 0.8588) demonstrated the strongest performance. Basically allthe models had almost similar performance except for The Deep Learning Classifier which showed comparatively weaker performance in terms of F1-score.

C. Clustering Insights

Phase 4 also explored clustering algorithms (KMeans and DBSCAN) for pattern discovery. K-Means analysis with $k = 2$ was found to be optimal, identifying two distinct clusters. Notably, these clusters exhibited different fare distributions, with Cluster 1 associated with significantly higher average fares than Cluster 0. While not direct predictive models, these clustering insights can be

valuable for deeper data understanding, potentially informing feature engineering for specific customer segments or fare structures in future iterations. The PCA visualizations (Fig. 22) confirmed distinct cluster formations.

D. Overall Best Model Selection and Justification

For the Regression Task (Predicting 'fare_amount'): Based on the reported R^2 values and error metrics (where available and consistent), the **Gradient Boosting Regressor** stands out as the most performant model. The **DecisionTree Regressor** is also a very strong candidate. These models effectively capture the relationship between the features and the continuous fare amount.

For the Classification Task (Predicting Fare Ranges): The **Bagging Classifier** and the **kNN Classifier** are the top performers. They demonstrate high accuracy and robust F1-scores, indicating a good balance of precision and recall in assigning trips to fare categories.

E. Considerations and Reccomendations:

The choice of the "best" model can also depend on factors not fully detailed in the current metrics, such as training time, inference speed, and interpretability. Ensemble models like Gradient Boosting and Bagging are often computationally more intensive but yield high accuracy. kNN can be computationally expensive at prediction time for large datasets if not efficiently implemented (though the use of BallTree mitigates this. Decision Trees offer high interpretability, but their standalone performance was generally lower than ensemble methods or kNN in this project. Deep Learning models have the potential for high performance but may require more extensive tuning and larger and complex datasets to reach their full potential, and in this case for this simple dataset their reported F1 score for classification was lower than other methods. I would lastly suggest to further explore what caused the poor performance for MLPRegressor and DeepeLearningRegressor models on Regression.

VI. Operationalisation

Deploying the machine learning model for predicting taxi fare amounts in production requires a robust pipeline that ensures scalability, reliability, and maintainability. The following components are essential:

A. Model Serving

The trained model can be containerised using Docker and deployed via a REST API using frameworks like Flask or FastAPI. This allows the model to serve real-time predictions by receiving trip features (e.g., pickup/dropoff coordinates, datetime) and returning fare predictions.

B. Data Ingestion

In production, trip data will be collected in real-time or batch via data streaming platforms such as Apache Kafka or ingestion APIs. A pre-processing layer must ensure incoming data matches the expected schema, handles missing or malformed inputs, and applies the same transformations used during training.

C. Monitoring and Logging

Model performance should be continuously monitored to detect data drift or degradation in predictive accuracy. This includes logging prediction requests, response times, and model confidence scores. Alerting mechanisms can notify when anomalies occur, e.g., sudden spikes in prediction errors.

D. Model Retraining

A retraining pipeline should be in place to update the model periodically using new trip data. This can be automated using workflow orchestration tools such as Apache Airflow or Prefect, ensuring the model remains accurate over time.

E. Scalability and Reliability

To support high volumes of requests, the deployment should be horizontally scalable using container orchestration platforms like Kubernetes. Load balancing and redundancy are crucial to maintain availability and fault tolerance.

F. Security and Compliance

Access to the model API must be secured using authentication (e.g., OAuth2) and HTTPS. Data privacy regulations (e.g., GDPR) must be considered, especially when handling location and customer data.

By addressing these components, the deployed machine learning system can provide reliable and timely fare predictions, improving the efficiency and transparency of the taxi service.

VII. Conclusion

This report presented the initial phases of a machine learning project aimed at predicting taxi fare amounts in New York City. Beginning with a well-defined problem formulation, we performed extensive data preprocessing, including data cleaning, outlier removal, and feature engineering, to prepare the dataset for modeling. Exploratory Data Analysis (EDA) helped uncover key relationships and patterns, such as the strong correlation between fare amount, trip distance, and trip duration.

Multiple models were evaluated for both regression and classification tasks. Among them, Gradient Boosting Regressor achieved the highest performance for predicting continuous fare amounts, while the Bagging Classifier and kNN Classifier demonstrated strong results in classifying fare ranges. Deep learning models, while promising, showed mixed performance and may require more tuning for optimal results.

The operationalisation section outlined key considerations for deploying the selected models in production, ensuring real-time inference, data integrity, monitoring, and scalability.

Overall, this project established a solid foundation for taxi fare prediction using data-driven techniques. Future work can focus on incorporating external data sources (e.g., weather, traffic), improving model generalizability, and integrating the model into real-world applications to enhance user experience and operational efficiency in urban mobility services.

TABLE III: Fold Evaluations for KNNRegressor (Regression)

Fold ID	Best Params	MSE	RMSE	MAE	R2 Score
1	'k': 7	4.47e-05	0.00669	0.00120	0.9664
2	'k': 7	3.36e-05	0.00580	0.00115	0.9739
3	'k': 7	3.70e-05	0.00608	0.00117	0.9713
4	'k': 7	3.09e-05	0.00555	0.00117	0.9764
5	'k': 7	2.45e-05	0.00495	0.00113	0.9816

TABLE IV: Fold Evaluations for KNNClassifier (Classification)

Fold ID	Best Params	Accuracy	Precision	Recall	F1 Score
1	'k': 7	0.9827	0.9366	0.8974	0.9148
2	'k': 7	0.9840	0.9657	0.9126	0.9353
3	'k': 7	0.9840	0.9490	0.9011	0.9219
4	'k': 7	0.9825	0.9326	0.8929	0.9106
5	'k': 7	0.9831	0.9723	0.9263	0.9467

TABLE V: Fold Evaluations for DecisionTreeRegressor (Regression)

Fold ID	Params Tried	MSE	RMSE	MAE	R2 Score
1	{'criterion': 'squared_error', 'max_depth': 7, 'max_features': 0.7, 'min_samples_leaf': 5, 'min_samples_split': 20}	4.79e-05	0.00692	0.00273	0.9625
2	{'criterion': 'squared_error', 'max_depth': 7, 'max_features': 0.7, 'min_samples_leaf': 5, 'min_samples_split': 20}	5.81e-05	0.00763	0.00277	0.9574
3	{'criterion': 'squared_error', 'max_depth': 7, 'max_features': 0.7, 'min_samples_leaf': 5, 'min_samples_split': 20}	3.37e-05	0.00580	0.00265	0.9745
4	{'criterion': 'squared_error', 'max_depth': 7, 'max_features': 0.7, 'min_samples_leaf': 5, 'min_samples_split': 20}	4.50e-05	0.00671	0.00246	0.9663
5	{'criterion': 'squared_error', 'max_depth': 7, 'max_features': 0.7, 'min_samples_leaf': 5, 'min_samples_split': 20}	4.26e-05	0.00653	0.00240	0.9660

TABLE VI: Fold Evaluations for DecisionTreeClassifier (Classification)

Fold ID	Params Tried	Accuracy	Precision	Recall	F1 Score
1	{'max_depth': 10, 'min_samples_leaf': 2, 'min_samples_split': 5}	0.9813	0.9292	0.8697	0.8944
2	{'max_depth': 10, 'min_samples_leaf': 2, 'min_samples_split': 5}	0.9833	0.9531	0.8948	0.9193
3	{'max_depth': 10, 'min_samples_leaf': 2, 'min_samples_split': 5}	0.9832	0.9396	0.9245	0.9317
4	{'max_depth': 10, 'min_samples_leaf': 2, 'min_samples_split': 5}	0.9823	0.9495	0.8924	0.9165
5	{'max_depth': 10, 'min_samples_leaf': 2, 'min_samples_split': 5}	0.9835	0.9519	0.9040	0.9250

TABLE VII: Fold Evaluations for MLPRegressor (Regression)

Fold ID	Params Tried	MSE	RMSE	MAE	R2 Score
1	{'activation': 'relu', 'alpha': 0.01, 'early_stopping': True, 'hidden_layer_sizes': (50,), 'learning_rate_init': 0.001, 'n_iter_no_change': 10, 'validation_fraction': 0.2}	3.68e-05	0.00607	0.00146	0.9719
2	{'activation': 'relu', 'alpha': 0.01, 'early_stopping': True, 'hidden_layer_sizes': (50,), 'learning_rate_init': 0.001, 'n_iter_no_change': 10, 'validation_fraction': 0.2}	3.76e-05	0.00613	0.00149	0.9714
3	{'activation': 'relu', 'alpha': 0.01, 'early_stopping': True, 'hidden_layer_sizes': (50,), 'learning_rate_init': 0.001, 'n_iter_no_change': 10, 'validation_fraction': 0.2}	3.63e-05	0.00602	0.00152	0.9718
4	{'activation': 'relu', 'alpha': 0.01, 'early_stopping': True, 'hidden_layer_sizes': (50,), 'learning_rate_init': 0.001, 'n_iter_no_change': 10, 'validation_fraction': 0.2}	3.33e-05	0.00577	0.00166	0.9741
5	{'activation': 'relu', 'alpha': 0.01, 'early_stopping': True, 'hidden_layer_sizes': (50,), 'learning_rate_init': 0.001, 'n_iter_no_change': 10, 'validation_fraction': 0.2}	2.56e-05	0.00506	0.00140	0.9800

TABLE VIII: Fold Evaluations for MLPClassifier (Classification)

Fold ID	Params Tried	Accuracy	Precision	Recall	F1 Score
1	{'activation': 'tanh', 'alpha': 0.0001, 'early_stopping': True, 'hidden_layer_sizes': (50, 25), 'learning_rate_init': 0.001}	0.9832	0.9602	0.8875	0.9169
2	{'activation': 'tanh', 'alpha': 0.0001, 'early_stopping': True, 'hidden_layer_sizes': (50, 25), 'learning_rate_init': 0.001}	0.9847	0.9391	0.9100	0.9231
3	{'activation': 'tanh', 'alpha': 0.0001, 'early_stopping': True, 'hidden_layer_sizes': (50, 25), 'learning_rate_init': 0.001}	0.9845	0.9310	0.9657	0.9468
4	{'activation': 'tanh', 'alpha': 0.0001, 'early_stopping': True, 'hidden_layer_sizes': (50, 25), 'learning_rate_init': 0.001}	0.9841	0.9497	0.9206	0.9341
5	{'activation': 'tanh', 'alpha': 0.0001, 'early_stopping': True, 'hidden_layer_sizes': (50, 25), 'learning_rate_init': 0.001}	0.9845	0.9553	0.8884	0.9152

TABLE IX: Fold Evaluations for GradientBoostingRegressor (Regression)

Fold ID	Params Tried	MSE	RMSE	MAE	R2 Score
1	{'learning_rate': 0.1, 'max_depth': 3, 'max_features': 0.5, 'n_estimators': 100, 'subsample': 0.8}	2.82e-05	0.00531	0.00148	0.9780
2	{'learning_rate': 0.1, 'max_depth': 3, 'max_features': 0.5, 'n_estimators': 100, 'subsample': 0.8}	4.24e-05	0.00651	0.00146	0.9689

Continued on next page

TABLE IX: (Continued)

Fold ID	Params Tried	MSE	RMSE	MAE	R2 Score
3	{'learning_rate': 0.1, 'max_depth': 3, 'max_features': 0.5, 'n_estimators': 100, 'subsample': 0.8}	2.06e-05	0.00454	0.00143	0.9843
4	{'learning_rate': 0.1, 'max_depth': 3, 'max_features': 0.5, 'n_estimators': 100, 'subsample': 0.8}	3.52e-05	0.00594	0.00149	0.9736
5	{'learning_rate': 0.1, 'max_depth': 3, 'max_features': 0.5, 'n_estimators': 100, 'subsample': 0.8}	2.71e-05	0.00521	0.00145	0.9783

TABLE X: Fold Evaluations for GradientBoostingClassifier (Classification)

Fold ID	Params Tried	Accuracy	Precision	Recall	F1 Score
1	{'learning_rate': 0.05, 'max_depth': 5, 'n_estimators': 100}	0.9834	0.9264	0.8972	0.9106
2	{'learning_rate': 0.05, 'max_depth': 5, 'n_estimators': 100}	0.9847	0.9141	0.9391	0.9258
3	{'learning_rate': 0.05, 'max_depth': 5, 'n_estimators': 100}	0.9845	0.9334	0.9565	0.9443
4	{'learning_rate': 0.05, 'max_depth': 5, 'n_estimators': 100}	0.9841	0.9484	0.9039	0.9233
5	{'learning_rate': 0.05, 'max_depth': 5, 'n_estimators': 100}	0.9853	0.9549	0.9337	0.9437

TABLE XI: Fold Evaluations for DeepLearningRegressor (Regression)

Fold ID	Params Tried	MSE	RMSE	MAE	R2 Score
1	{'units_layer1': 64, 'batch_size': 32, 'learning_rate': 0.0001}	0.00010	0.01019	0.00464	0.9207
2	{'units_layer1': 64, 'batch_size': 32, 'learning_rate': 0.0001}	0.00010	0.01021	0.00468	0.9201
3	{'units_layer1': 64, 'batch_size': 32, 'learning_rate': 0.0001}	0.00012	0.01075	0.00449	0.9112
4	{'units_layer1': 64, 'batch_size': 32, 'learning_rate': 0.0001}	0.00010	0.01007	0.00473	0.9199
5	{'units_layer1': 64, 'batch_size': 32, 'learning_rate': 0.0001}	0.00017	0.01304	0.00797	0.8690

TABLE XII: Fold Evaluations for DeepLearningClassifier (Classification)

Fold ID	Params Tried	Accuracy	Precision	Recall	F1 Score
1	{'units_layer1': 64, 'batch_size': 64, 'learning_rate': 0.01}	0.9756	0.7108	0.7300	0.7200
2	{'units_layer1': 64, 'batch_size': 64, 'learning_rate': 0.01}	0.9708	0.7255	0.7017	0.7129
3	{'units_layer1': 64, 'batch_size': 64, 'learning_rate': 0.01}	0.9771	0.7218	0.7269	0.7242
4	{'units_layer1': 64, 'batch_size': 64, 'learning_rate': 0.01}	0.9746	0.7246	0.7053	0.7144
5	{'units_layer1': 64, 'batch_size': 64, 'learning_rate': 0.01}	0.9722	0.7230	0.6940	0.7067

TABLE XIII: Fold Evaluations for BaggingRegressor (Regression)

Fold ID	Params Tried	MSE	RMSE	MAE	R2 Score
1	'bootstrap': True, 'bootstrap_features': True, 'max_features': 0.7, 'max_samples': 0.5, 'n_estimators': 50	$8.35e-05$	0.00914	0.00522	0.9347
2	'bootstrap': True, 'bootstrap_features': True, 'max_features': 0.7, 'max_samples': 0.5, 'n_estimators': 50	$1.07e-04$	0.01034	0.00523	0.9216
3	'bootstrap': True, 'bootstrap_features': True, 'max_features': 0.7, 'max_samples': 0.5, 'n_estimators': 50	$8.19e-05$	0.00905	0.00518	0.9379
4	'bootstrap': True, 'bootstrap_features': True, 'max_features': 0.7, 'max_samples': 0.5, 'n_estimators': 50	$9.45e-05$	0.00972	0.00522	0.9291
5	'bootstrap': True, 'bootstrap_features': True, 'max_features': 0.7, 'max_samples': 0.5, 'n_estimators': 50	$8.24e-05$	0.00908	0.00521	0.9342

TABLE XIV: Fold Evaluations for BaggingClassifier (Classification)

Fold ID	Params Tried	Accuracy	Precision	Recall	F1 Score
1	'bootstrap': True, 'bootstrap_features': False, 'max_samples': 0.5, 'n_estimators': 50	0.9819	0.9516	0.9053	0.9258
2	'bootstrap': True, 'bootstrap_features': False, 'max_samples': 0.5, 'n_estimators': 50	0.9828	0.9417	0.9230	0.9318
3	'bootstrap': True, 'bootstrap_features': False, 'max_samples': 0.5, 'n_estimators': 50	0.9830	0.9479	0.9523	0.9501
4	'bootstrap': True, 'bootstrap_features': False, 'max_samples': 0.5, 'n_estimators': 50	0.9823	0.9634	0.9110	0.9337
5	'bootstrap': True, 'bootstrap_features': False, 'max_samples': 0.5, 'n_estimators': 50	0.9832	0.9582	0.9224	0.9388

REFERENCES

[1] Dhruvil Dave, "New York City Taxi Trips 2019 Dataset," Kaggle. Available: https://www.kaggle.com/datasets/dhruvildave/new-york-city-taxi-trips-2019/data. Accessed: May 26, 2025.