Taxi Fare Prediction Modeling

Introduction

Research Objective

The primary objective of this study is to develop a predictive fare model using the comprehensive NYC yellow taxi trip dataset from 2023. The aim is to identify key fare determinants and leverage advanced machine learning techniques to create a model that can estimate taxi fares accurately. This model will then be adapted to propose a more structured and transparent fare system for Tbilisi, Georgia.

Summary of EDA Insights and Hypotheses

From the exploratory data analysis, several key insights were derived:

- The fare amount, trip distance, trip duration, speed, and tip amount distributions were highly right-skewed, indicating the need for transformations.
- Fare amount had strong positive correlations with trip distance and trip duration.
- Categorical variables such as time of day, day type, season, and holidays significantly impacted fare amounts.
- Proper handling of outliers was crucial to prevent skewing model predictions.

Hypotheses

1. **Trip Distance and Duration:** Longer trips (both in terms of distance and duration) will result in higher fares.
2. **Temporal Factors:** Taxi fares vary significantly by time of day, day of the week, and season, with peak times and special events (e.g., holidays) resulting in higher fares.
3. **Speed:** Higher average speeds may correlate with higher fares due to longer distances covered in shorter times.
4. **Tips:** Higher fares tend to receive higher tips.

The modeling phase will test these hypotheses by evaluating the predictive power of these factors.

Methodology

Data Preparation

The dataset used for modeling has been cleaned and preprocessed based on the insights from the EDA. Key steps include:

- Log transformations to normalize skewed data.
- Handling outliers using Z-score filtering.
- Encoding categorical variables using one-hot encoding.
- Scaling numerical features.
- PCA analysis

Model Selection

To build a robust fare prediction model, we will use both simple and complex machine learning models:

1. **Simple Models:**
    - Linear Regression
2. **Complex Models:**
    - Decision Trees
    - Random Forest
    - Gradient Boosting Machines (GBM)
    - XGBoost

Model Evaluation Metrics

The performance of the models will be evaluated using the following metrics:

- Mean Absolute Error (MAE)
- Mean Squared Error (MSE)
- R-squared (R^2)

These metrics will help in understanding the accuracy and reliability of the models.

In [42]:
```
# Import necessary libraries
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoo
import xgboost as xgb
from sklearn.metrics import mean_absolute_error, mean_squared_e
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoo
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
import xgboost as xgb
from sklearn.metrics import mean_absolute_error, mean_squared_e
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import GridSearchCV
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.model_selection import train_test_split, Randomize
```

In [9]:
```
# Load the prepared dataset
df = pd.read_parquet('modeling_dataset_v.0.parquet')
```

In [10]:
```
df.head()
```

Out[10]:

	fare_amount	trip_duration	trip_distance	total_amount	tip_amount	speed_mph
19276873	19.1	20.366667	2.32	30.72	5.12	6.834697
25505385	17.7	18.933333	1.90	25.20	3.50	6.021127
23056552	8.6	8.950000	1.10	12.10	2.00	7.374302
22171336	10.0	8.966667	1.00	16.00	2.00	6.691450
25241061	7.9	5.783333	1.07	14.88	2.98	11.100865

5 rows × 24 columns

In [11]:
```
len(df)
```

Out[11]:
```
2805668
```

In [12]:
```
df.dtypes
```

Out[12]:
```
fare_amount                 float64
trip_duration               float64
trip_distance               float64
total_amount                float64
tip_amount                  float64
speed_mph                   float64
JFK_LGA_Pickup_Fee          float64
General_Airport_Fee         float64
log_fare_amount             float64
log_trip_duration           float64
log_trip_distance           float64
log_tip_amount              float64
is_holiday                    int64
payment_type                  int64
pickup_time_of_day         category
pickup_season              category
passenger_count_category   category
pickup_day_type            category
PUzone                     category
PUborough                  category
DOzone                     category
DOborough                  category
PCA1                        float64
PCA2                        float64
dtype: object
```

In [13]:

```


# Create a combined column for stratification
df['stratify_col'] = df['pickup_season'].astype(str) + '_' + df

# Check the distribution of the combined stratification column
stratify_counts = df['stratify_col'].value_counts()
print(stratify_counts)

# Filter out classes with very few samples
min_class_size = 5  # Define a minimum class size
filtered_classes = stratify_counts[stratify_counts >= min_class
df_filtered = df[df['stratify_col'].isin(filtered_classes)]

# Perform stratified sampling on the filtered dataset
df_sample, _ = train_test_split(df_filtered, test_size=0.9, str

# Drop the stratify column
df_sample = df_sample.drop(columns=['stratify_col'])

```

```
stratify_col
spring_evening_weekday_Manhattan_Manhattan      162096
spring_afternoon_weekday_Manhattan_Manhattan    152673
autumn_evening_weekday_Manhattan_Manhattan      149395
autumn_afternoon_weekday_Manhattan_Manhattan    140089
winter_afternoon_weekday_Manhattan_Manhattan    139463
                                                  ...
autumn_night_weekend_Unknown_EWR                     1
autumn_afternoon_weekday_EWR_EWR                     1
autumn_evening_weekday_Unknown_EWR                   1
spring_night_weekday_Staten Island_Queens            1
winter_night_weekday_Unknown_EWR                     1
Name: count, Length: 1050, dtype: int64
```

In [23]:

```
print(len(df))
print(len(df_sample))
```

```
2805668
280513
```

For the modeling purposes we performed stratified sampling to ensure that all the categorical features remain intact and we have a model that can give us insights based on the categorical variables as well.

In [ ]:

```
# Feature selection and encoding
features = df_sample[['PCA1', 'PCA2', 'is_holiday', 'pickup_tir
target = df_sample['fare_amount']

categorical_features = ['is_holiday', 'pickup_time_of_day', 'p:
encoder = ColumnTransformer(transformers=[('cat', OneHotEncode
```

```

X = encoder.fit_transform(features)
y = target.values

# Reduce the number of features using SelectKBest
X_new = SelectKBest(f_regression, k=10).fit_transform(X, y)

# Split data into training, validation, and test sets
X_train, X_temp, y_train, y_temp = train_test_split(X_new, y, 
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp

# Standardize the data for KNN and SVM
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
X_test_scaled = scaler.transform(X_test)

# Linear Regression
linear_model = LinearRegression()
linear_model.fit(X_train, y_train)
y_pred_train_lr = linear_model.predict(X_train)
y_pred_val_lr = linear_model.predict(X_val)
print("Linear Regression Performance")
print("Training MAE:", mean_absolute_error(y_train, y_pred_tra
print("Validation MAE:", mean_absolute_error(y_val, y_pred_val_
print("Validation R²:", r2_score(y_val, y_pred_val_lr))

# Decision Tree with Grid Search for parameter tuning
param_grid_dt = {'max_depth': [None, 10, 20], 'min_samples_spl
dt_grid_search = GridSearchCV(DecisionTreeRegressor(random_sta
dt_grid_search.fit(X_train, y_train)
best_dt_model = dt_grid_search.best_estimator_
y_pred_val_dt = best_dt_model.predict(X_val)
print("Decision Tree Performance")
print("Validation MAE:", mean_absolute_error(y_val, y_pred_val_
print("Validation R²:", r2_score(y_val, y_pred_val_dt))

# Random Forest with Grid Search for parameter tuning
param_grid_rf = {'n_estimators': [10, 50, 100], 'max_depth': [
rf_grid_search = GridSearchCV(RandomForestRegressor(random_sta
rf_grid_search.fit(X_train, y_train)
best_rf_model = rf_grid_search.best_estimator_
y_pred_val_rf = best_rf_model.predict(X_val)
print("Random Forest Performance")
print("Validation MAE:", mean_absolute_error(y_val, y_pred_val_
print("Validation R²:", r2_score(y_val, y_pred_val_rf))

# Gradient Boosting with early stopping
gb_model = GradientBoostingRegressor(n_estimators=100, validat
gb_model.fit(X_train, y_train)
y_pred_val_gb = gb_model.predict(X_val)
print("Gradient Boosting Performance")
print("Validation MAE:", mean_absolute_error(y_val, y_pred_val_
```

```
print("Validation R²:", r2_score(y_val, y_pred_val_gb))

# XGBoost with Grid Search for parameter tuning
param_grid_xgb = {'n_estimators': [50, 100], 'max_depth': [3, 
xgb_grid_search = GridSearchCV(xgb.XGBRegressor(random_state=42
xgb_grid_search.fit(X_train, y_train)
best_xgb_model = xgb_grid_search.best_estimator_
y_pred_val_xgb = best_xgb_model.predict(X_val)
print("XGBoost Performance")
print("Validation MAE:", mean_absolute_error(y_val, y_pred_val_
print("Validation R²:", r2_score(y_val, y_pred_val_xgb))

# K-Nearest Neighbors
knn_model = KNeighborsRegressor(n_neighbors=5, n_jobs=-1)
knn_model.fit(X_train_scaled, y_train)
y_pred_val_knn = knn_model.predict(X_val_scaled)
print("KNN Performance")
print("Validation MAE:", mean_absolute_error(y_val, y_pred_val_
print("Validation R²:", r2_score(y_val, y_pred_val_knn))

# Support Vector Machine
svm_model = SVR(kernel='rbf')
svm_model.fit(X_train_scaled, y_train)
y_pred_val_svm = svm_model.predict(X_val_scaled)
print("SVM Performance")
print("Validation MAE:", mean_absolute_error(y_val, y_pred_val_
print("Validation R²:", r2_score(y_val, y_pred_val_svm))

# Summarize Model Performance
models = ['Linear Regression', 'Decision Tree', 'Random Forest
val_mae = [
    mean_absolute_error(y_val, y_pred_val_lr),
    mean_absolute_error(y_val, y_pred_val_dt),
    mean_absolute_error(y_val, y_pred_val_rf),
    mean_absolute_error(y_val, y_pred_val_gb),
    mean_absolute_error(y_val, y_pred_val_xgb),
    mean_absolute_error(y_val, y_pred_val_knn),
    mean_absolute_error(y_val, y_pred_val_svm)
]
val_r2 = [
    r2_score(y_val, y_pred_val_lr),
    r2_score(y_val, y_pred_val_dt),
    r2_score(y_val, y_pred_val_rf),
    r2_score(y_val, y_pred_val_gb),
    r2_score(y_val, y_pred_val_xgb),
    r2_score(y_val, y_pred_val_knn),
    r2_score(y_val, y_pred_val_svm)
]

performance_df = pd.DataFrame({
    'Model': models,
    'Validation MAE': val_mae,
    'Validation R²': val_r2
```

```
})

print(performance_df)
```

Preparation For Modeling

In [29]:
```
# Feature selection and encoding
features = df_sample[['PCA1', 'PCA2', 'is_holiday', 'pickup_tim
target = df_sample['fare_amount']
```

In [30]:
```
categorical_features = ['is_holiday', 'pickup_time_of_day', 'pi
encoder = ColumnTransformer(transformers=[('cat', OneHotEncoder
```

In [32]:
```
X = encoder.fit_transform(features)
y = target.values

# Reduce the number of features using SelectKBest
X_new = SelectKBest(f_regression, k=10).fit_transform(X, y)

# Split data into training, validation, and test sets
X_train, X_temp, y_train, y_temp = train_test_split(X_new, y, t
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp,

```

In [35]:
```
# Standardize the data for KNN and SVM
scaler = StandardScaler(with_mean= False)
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
X_test_scaled = scaler.transform(X_test)

```

Baseline Model

In [36]:
```
# Baseline Model: Linear Regression
linear_model = LinearRegression()
linear_model.fit(X_train, y_train)
y_pred_train = linear_model.predict(X_train)
y_pred_val = linear_model.predict(X_val)

print("Linear Regression Performance")
print("Training MAE:", mean_absolute_error(y_train, y_pred_trai
print("Validation MAE:", mean_absolute_error(y_val, y_pred_val)
print("Validation R²:", r2_score(y_val, y_pred_val))

```

```
Linear Regression Performance
Training MAE: 2.697683248247949
Validation MAE: 2.6931883885075365
Validation R²: 0.944835876489503
```

Linear Regression Model

Performance Metrics:

- **Training MAE:** 2.70
- **Validation MAE:** 2.69
- **Validation R^2:** 0.9448

Interpretation:

1. **Mean Absolute Error (MAE):**

    - The average absolute error in fare prediction is approximately $2.70 on the training data and $2.69 on the validation data. The closeness of these values indicates that the model is not overfitting and generalizes well to unseen data.

2. **R-squared (R^2):**

    - The R^2 value of 0.9448 indicates that approximately 94.48% of the variance in taxi fares can be explained by the model. This high R^2 value suggests that the linear regression model fits the data well.

Conclusion:

- The linear regression model performs well with a high R^2 value and low MAE, indicating strong predictive capability for the given features. However, it is a simple model and may not capture complex, non-linear relationships in the data.

Complex Models

In [39]:
```

# Decision Tree with Grid Search for parameter tuning
param_grid_dt = {'max_depth': [None, 10, 20], 'min_samples_spli
dt_grid_search = GridSearchCV(DecisionTreeRegressor(random_stat
dt_grid_search.fit(X_train, y_train)
best_dt_model = dt_grid_search.best_estimator_
y_pred_val_dt = best_dt_model.predict(X_val)
print("Decision Tree Performance")
print("Validation MAE:", mean_absolute_error(y_val, y_pred_val_
print("Validation R²:", r2_score(y_val, y_pred_val_dt))

```

```
Decision Tree Performance
Validation MAE: 2.3716119571271497
Validation R²: 0.9545403292306762
```

Decision Tree Model

Performance Metrics:

- **Validation MAE:** 2.37
- **Validation R^2:** 0.9545

Interpretation:

1. **Mean Absolute Error (MAE):**
   - The average absolute error in fare prediction is approximately $2.37 on the validation data. This is slightly lower than the MAE of the linear regression model, indicating better predictive accuracy.
2. **R-squared (R^2):**
   - The R^2 value of 0.9545 indicates that approximately 95.45% of the variance in taxi fares can be explained by the decision tree model. This is higher than the R^2 value for the linear regression model, suggesting that the decision tree captures more variability in the data.

Conclusion:

- The decision tree model performs better than the linear regression model in terms of both MAE and R^2. It captures more complex relationships between the features and the target variable. However, decision trees can be prone to overfitting, so it is essential to validate the model on unseen data and potentially prune the tree or limit its depth to prevent overfitting.

In [44]:
```
# Gradient Boosting with early stopping
gb_model = GradientBoostingRegressor(n_estimators=100, validati
gb_model.fit(X_train, y_train)
y_pred_val_gb = gb_model.predict(X_val)
print("Gradient Boosting Performance")
print("Validation MAE:", mean_absolute_error(y_val, y_pred_val_
print("Validation R²:", r2_score(y_val, y_pred_val_gb))

```

```
Gradient Boosting Performance
Validation MAE: 2.41214002650482
Validation R²: 0.9550262508155019
```

In [45]:
```
# XGBoost with Grid Search for parameter tuning
param_grid_xgb = {'n_estimators': [50, 100], 'max_depth': [3, 6
xgb_grid_search = GridSearchCV(xgb.XGBRegressor(random_state=42
xgb_grid_search.fit(X_train, y_train)
best_xgb_model = xgb_grid_search.best_estimator_
y_pred_val_xgb = best_xgb_model.predict(X_val)
print("XGBoost Performance")
print("Validation MAE:", mean_absolute_error(y_val, y_pred_val_
print("Validation R²:", r2_score(y_val, y_pred_val_xgb))
```

```
XGBoost Performance
Validation MAE: 2.3678238791443933
Validation R²: 0.9528451666259213
```

In [ ]:
```
# Random Forest with Randomized Search for parameter tuning usi
param_dist_rf = {'n_estimators': [10, 50], 'max_depth': [10, 20
X_train_sample, _, y_train_sample, _ = train_test_split(X_train
rf_random_search = RandomizedSearchCV(RandomForestRegressor(ran
rf_random_search.fit(X_train_sample, y_train_sample)
best_rf_model = rf_random_search.best_estimator_
final_rf_model = RandomForestRegressor(**best_rf_model.get_para
final_rf_model.fit(X_train, y_train)
y_pred_val_rf = final_rf_model.predict(X_val)
print("Random Forest Performance")
print("Validation MAE:", mean_absolute_error(y_val, y_pred_val_
print("Validation R²:", r2_score(y_val, y_pred_val_rf))

```

In [47]:
```
# K-Nearest Neighbors
knn_model = KNeighborsRegressor(n_neighbors=5, n_jobs=-1)
knn_model.fit(X_train_scaled, y_train)
y_pred_val_knn = knn_model.predict(X_val_scaled)
print("KNN Performance")
print("Validation MAE:", mean_absolute_error(y_val, y_pred_val_
print("Validation R²:", r2_score(y_val, y_pred_val_knn))

```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent c
```

```
KeyboardInterrupt                         Traceback (most recent c
all last)
/var/folders/4d/8tkcz58x0md0_v7fj12j6ht80000gn/T/ipykernel_2648/17
13529712.py in <module>
      2 knn_model = KNeighborsRegressor(n_neighbors=5, n_jobs=-1)
      3 knn_model.fit(X_train_scaled, y_train)
----> 4 y_pred_val_knn = knn_model.predict(X_val_scaled)
      5 print("KNN Performance")
      6 print("Validation MAE:", mean_absolute_error(y_val,

y_pred_val_knn))

~/opt/anaconda3/lib/python3.9/site-packages/sklearn/neighbors/_reg
ression.py in predict(self, X)
    206         X = check_array(X, accept_sparse='csr')
    207
--> 208         neigh_dist, neigh_ind = self.kneighbors(X)
    209
    210         weights = _get_weights(neigh_dist, self.weights)

~/opt/anaconda3/lib/python3.9/site-packages/sklearn/neighbors/_bas
e.py in kneighbors(self, X, n_neighbors, return_distance)
    703                 kwds = self.effective_metric_params_
    704
--> 705             chunked_results = list(pairwise_distances_chun
ked(
    706                 X, self._fit_X, reduce_func=reduce_func,
    707                 metric=self.effective_metric_, n_jobs=n_jo
bs,

~/opt/anaconda3/lib/python3.9/site-packages/sklearn/metrics/pairwi
se.py in pairwise_distances_chunked(X, Y, reduce_func, metric, n_j
obs, working_memory, **kwds)
   1631         if reduce_func is not None:
   1632             chunk_size = D_chunk.shape[0]
-> 1633             D_chunk = reduce_func(D_chunk, sl.start)
   1634             _check_chunk_size(D_chunk, chunk_size)
   1635         yield D_chunk

~/opt/anaconda3/lib/python3.9/site-packages/sklearn/neighbors/_bas
e.py in _kneighbors_reduce_func(self, dist, start, n_neighbors, re
turn_distance)
    580         """
    581         sample_range = np.arange(dist.shape[0])[:, None]
--> 582         neigh_ind = np.argpartition(dist, n_neighbors - 1,
axis=1)
    583         neigh_ind = neigh_ind[:, :n_neighbors]
    584         # argpartition doesn't guarantee sorted order, so
we sort again

~/opt/anaconda3/lib/python3.9/site-packages/numpy/core/fromnumeric
.py in argpartition(a, kth, axis, kind, order)
    856
    857     """
--> 858     return _wrapfunc(a, 'argpartition', kth, axis=axis,
```

```
kind=kind, order=order)
    859
    860

~/opt/anaconda3/lib/python3.9/site-packages/numpy/core/fromnumeric
.py in _wrapfunc(obj, method, *args, **kwds)
     57
     58     try:

---> 59         return bound(*args, **kwds)
     60     except TypeError:
     61         # A TypeError occurs if the object does have such
a method in its

KeyboardInterrupt:
```

In [ ]:

```python
# Support Vector Machine
svm_model = SVR(kernel='rbf')
svm_model.fit(X_train_scaled, y_train)
y_pred_val_svm = svm_model.predict(X_val_scaled)
print("SVM Performance")
print("Validation MAE:", mean_absolute_error(y_val, y_pred_val_
print("Validation R²:", r2_score(y_val, y_pred_val_svm))

```

In [ ]:

```python
# Neural Network (MLP)
model = Sequential()
model.add(Dense(64, input_dim=X_train.shape[1], activation='rel
model.add(Dense(32, activation='relu'))
model.add(Dense(1))

# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
history = model.fit(X_train, y_train, epochs=50, batch_size=32,

# Predict on training and validation data
y_pred_train_nn = model.predict(X_train)
y_pred_val_nn = model.predict(X_val)

print("Neural Network Performance")
print("Training MAE:", mean_absolute_error(y_train, y_pred_trai
print("Validation MAE:", mean_absolute_error(y_val, y_pred_val_
print("Validation R²:", r2_score(y_val, y_pred_val_nn))

```

In [ ]:
```
models = ['Linear Regression', 'Decision Tree', 'Random Forest'
train_mae = [
    mean_absolute_error(y_train, linear_model.predict(X_train))
    mean_absolute_error(y_train, dt_model.predict(X_train)),
    mean_absolute_error(y_train, rf_model.predict(X_train)),
    mean_absolute_error(y_train, gb_model.predict(X_train)),
    mean_absolute_error(y_train, xgb_model.predict(X_train)),
    mean_absolute_error(y_train, knn_model.predict(X_train_scal
    mean_absolute_error(y_train, svm_model.predict(X_train_scal
    mean_absolute_error(y_train, model.predict(X_train))
]
val_mae = [
    mean_absolute_error(y_val, linear_model.predict(X_val)),
    mean_absolute_error(y_val, dt_model.predict(X_val)),
    mean_absolute_error(y_val, rf_model.predict(X_val)),
    mean_absolute_error(y_val, gb_model.predict(X_val)),
    mean_absolute_error(y_val, xgb_model.predict(X_val)),
    mean_absolute_error(y_val, knn_model.predict(X_val_scaled))
    mean_absolute_error(y_val, svm_model.predict(X_val_scaled))
    mean_absolute_error(y_val, model.predict(X_val))
]
val_r2 = [
    r2_score(y_val, linear_model.predict(X_val)),
    r2_score(y_val, dt_model.predict(X_val)),
    r2_score(y_val, rf_model.predict(X_val)),
    r2_score(y_val, gb_model.predict(X_val)),
    r2_score(y_val, xgb_model.predict(X_val)),
    r2_score(y_val, knn_model.predict(X_val_scaled)),
    r2_score(y_val, svm_model.predict(X_val_scaled)),
    r2_score(y_val, model.predict(X_val))
]

performance_df = pd.DataFrame({
    'Model': models,
    'Training MAE': train_mae,
    'Validation MAE': val_mae,
    'Validation R²': val_r2
})

print(performance_df)
```