Random Forest Regressor model 1

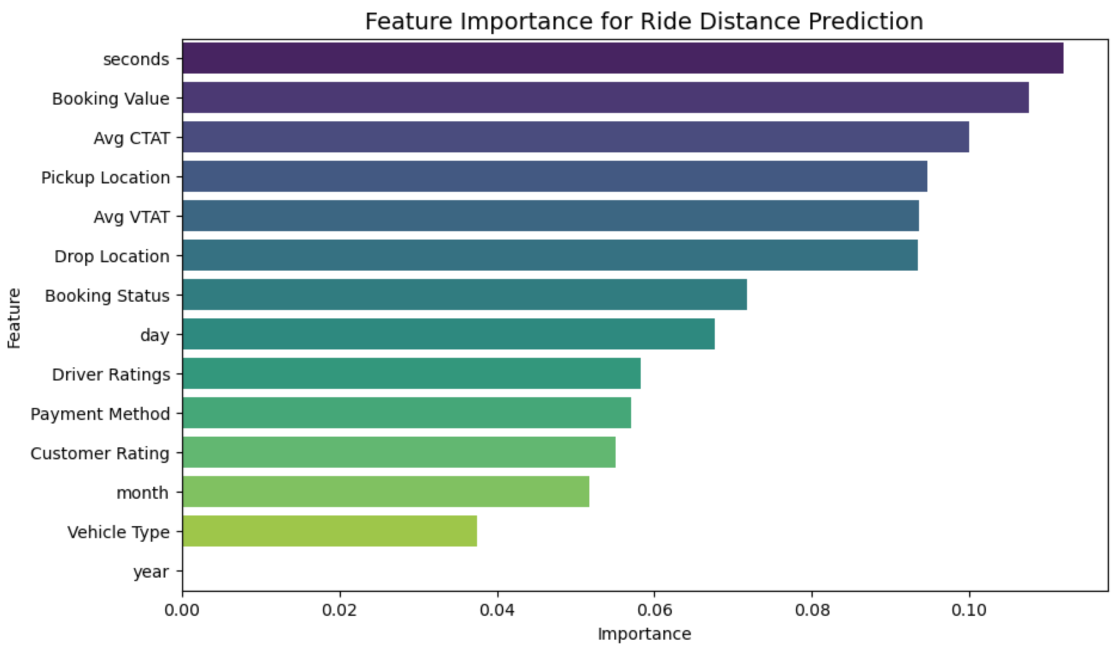
January 12, 2026

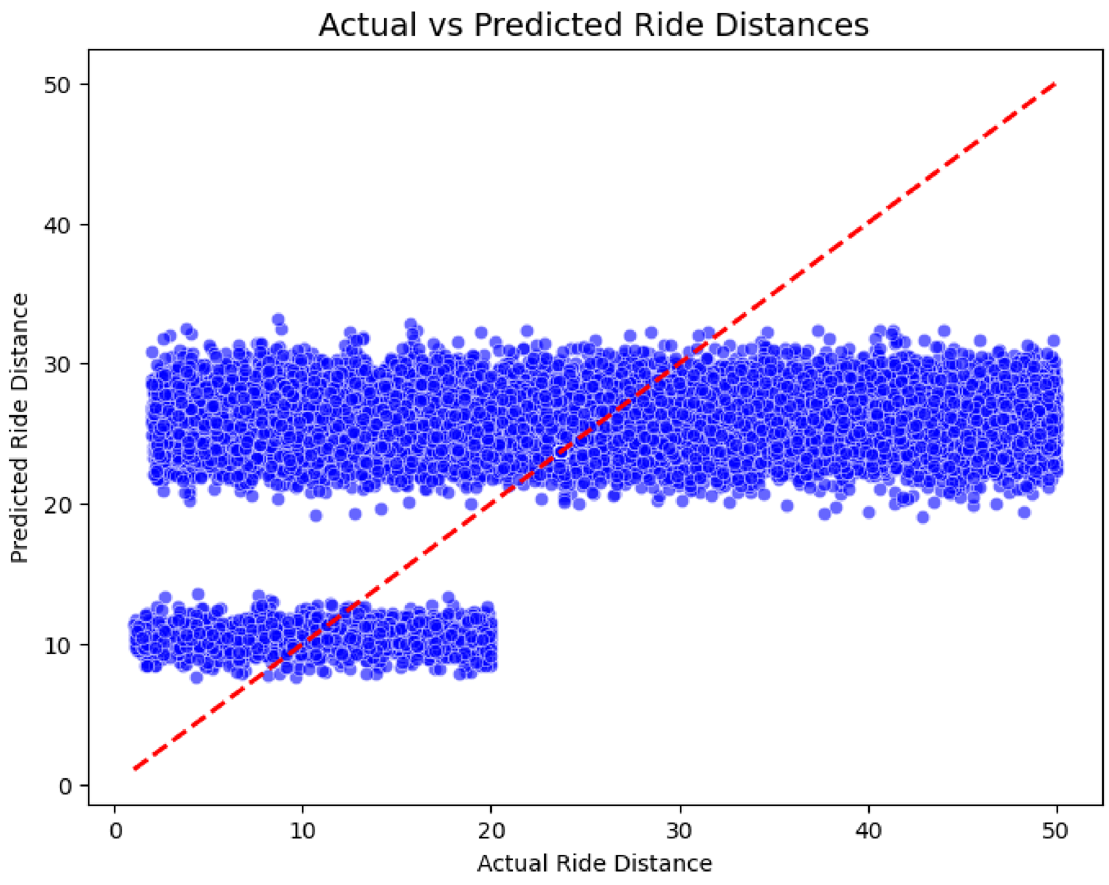
[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
from sklearn.ensemble import RandomForestRegressor

# Load dataset
df = pd.read_csv("ncr_ride_bookings.csv")

# Convert Date and Time
df['Date'] = pd.to_datetime(df['Date'])
df['year'] = df['Date'].dt.year
df['month'] = df['Date'].dt.month
df['day'] = df['Date'].dt.day

df['Time'] = pd.to_timedelta(df['Time'])
df['seconds'] = df['Time'].dt.total_seconds().astype(int)

# Select relevant columns
df = df[['year', 'month', 'day', 'seconds', 'Booking Status', 'Vehicle Type',
         'Pickup Location', 'Drop Location', 'Avg VTAT', 'Avg CTAT',
         'Booking Value', 'Ride Distance', 'Driver Ratings', 'Customer Rating',
         'Payment Method']]

# Handle missing values
df.fillna(df.median(numeric_only=True), inplace=True)

# Encode categorical variables
x_categorical = df.select_dtypes(include=['object']).apply(lambda col:␣
 ↪LabelEncoder().fit_transform(col))
x_numerical = df.select_dtypes(exclude=['object'])
X = pd.concat([x_numerical, x_categorical], axis=1).drop(columns=['Ride␣
 ↪Distance'])
y = df['Ride Distance'].values
```

```
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Random Forest Regressor
regressor = RandomForestRegressor(n_estimators=100, random_state=0)
regressor.fit(X_train, y_train)

# Predictions
predictions = regressor.predict(X_test)

# Evaluation
mse = mean_squared_error(y_test, predictions)
r2 = r2_score(y_test, predictions)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test, predictions)

print(f"Mean Squared Error: {mse:.2f}")
print(f"Root Mean Squared Error: {rmse:.2f}")
print(f"Mean Absolute Error: {mae:.2f}")
print(f"R-squared: {r2:.4f}")

# ========================
# Feature Importance Plot
# ========================
importances = regressor.feature_importances_
features = X.columns

feat_imp = pd.DataFrame({'Feature': features, 'Importance': importances})
feat_imp = feat_imp.sort_values(by='Importance', ascending=False)

plt.figure(figsize=(10,6))
sns.barplot(x='Importance', y='Feature', data=feat_imp, palette='viridis')
plt.title("Feature Importance for Ride Distance Prediction", fontsize=14)
plt.xlabel("Importance")
plt.ylabel("Feature")
plt.show()

# ========================
# Actual vs Predicted Scatter
# ========================
plt.figure(figsize=(8,6))
sns.scatterplot(x=y_test, y=predictions, alpha=0.6, color="blue")
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--',␣
 ↪lw=2)  # perfect prediction line
plt.title("Actual vs Predicted Ride Distances", fontsize=14)
plt.xlabel("Actual Ride Distance")
plt.ylabel("Predicted Ride Distance")
plt.show()
```

```
Mean Squared Error: 122.45
Root Mean Squared Error: 11.07
Mean Absolute Error: 7.74
R-squared: 0.0768
```

```
C:\Users\user\AppData\Local\Temp\ipykernel_5288\2725890830.py:68: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x='Importance', y='Feature', data=feat_imp, palette='viridis')
```




Actual vs Predicted Ride Distances
Predicted Ride Distance
Actual Ride Distance
0
10
20
30
40
50
0
10
20
30
40
50

```
[ ]:
```