In [14]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

df = pd.read_csv("uber.csv")
df.head()
```

Out[14]:

	Unnamed: 0	key	fare_amount	pickup_datetime	pickup_longitude	pick
0	24238194	2015-05-07 19:52:06.0000003	7.5	2015-05-07 19:52:06 UTC	-73.999817	
1	27835199	2009-07-17 20:04:56.0000002	7.7	2009-07-17 20:04:56 UTC	-73.994355	
2	44984355	2009-08-24 21:45:00.00000061	12.9	2009-08-24 21:45:00 UTC	-74.005043	
3	25894730	2009-06-26 08:22:21.0000001	5.3	2009-06-26 08:22:21 UTC	-73.976124	
4	17610152	2014-08-28 17:47:00.000000188	16.0	2014-08-28 17:47:00 UTC	-73.925023	

In [15]:
```
num_cols = df.select_dtypes(include=np.number).columns
df[num_cols] = df[num_cols].fillna(df[num_cols].median())

cat_cols = df.select_dtypes(exclude=np.number).columns
for col in cat_cols:
    df[col] = df[col].fillna(df[col].mode()[0])
```

In [16]:
```
cat_cols = df.select_dtypes(include="object").columns
cat_cols
```

Out[16]:
```
Index(['key', 'pickup_datetime'], dtype='object')
```

In [17]:
```
low_cardinality_cols = [
    col for col in cat_cols if df[col].nunique() <= 10
]

print("Encoding these columns:", low_cardinality_cols)

df = pd.get_dummies(df, columns=low_cardinality_cols, drop_first=True)
```

```
Encoding these columns: []
```

In [18]:
```
high_cardinality_cols = [
    col for col in cat_cols if df[col].nunique() > 50
]

print("Dropping columns:", high_cardinality_cols)

df.drop(columns=high_cardinality_cols, inplace=True)
```

```
Dropping columns: ['key', 'pickup_datetime']
```

In [19]:
```
X = df.iloc[:, :-1]
y = df.iloc[:, -1]
```

In [20]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.20,
    random_state=42)
```

In [21]:
```
X_train = X_train.select_dtypes(include=[np.number])
X_test = X_test.select_dtypes(include=[np.number])

print("Remaining dtypes in X_train:")
print(X_train.dtypes)
```

```
Remaining dtypes in X_train:
Unnamed: 0             int64
fare_amount          float64
pickup_longitude     float64
pickup_latitude      float64
dropoff_longitude    float64
dropoff_latitude     float64
dtype: object
```

In [22]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [23]:
```
corr_with_target = df.corr()[df.columns[-1]].sort_values(ascending=False)
corr_with_target
```

Out[23]:
```
passenger_count      1.000000
fare_amount          0.010150
Unnamed: 0           0.002257
dropoff_longitude    0.000034
pickup_longitude    -0.000414
dropoff_latitude    -0.000660
pickup_latitude     -0.001560
Name: passenger_count, dtype: float64
```

In [24]:
```
pd.DataFrame(X_train_scaled, columns=X_train.columns).to_csv(
    "X_train_scaled.csv", index=False
)

pd.DataFrame(X_test_scaled, columns=X_test.columns).to_csv(
    "X_test_scaled.csv", index=False
)
```

COMPONENT - 2

In [25]:
```
X_train = X_train.select_dtypes(include=[np.number])
X_test = X_test.select_dtypes(include=[np.number])
```

```
X_train.shape, X_test.shape
```

Out[25]: ((160000, 6), (40000, 6))

In [26]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [27]:
```
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import (
    mean_absolute_error,
    mean_squared_error,
    r2_score)
```

In [29]:
```
def evaluate_model(model, X_train, X_test, y_train, y_test):
    model.fit(X_train, y_train)
    preds = model.predict(X_test)

    mae = mean_absolute_error(y_test, preds)
    rmse = np.sqrt(mean_squared_error(y_test, preds))
    r2 = r2_score(y_test, preds)

    return mae, rmse, r2
```

In [30]:
```
df.columns
```

Out[30]:
```
Index(['Unnamed: 0', 'fare_amount', 'pickup_longitude', 'pickup_latitude',
       'dropoff_longitude', 'dropoff_latitude', 'passenger_count'],
      dtype='object')
```

In [31]:
```
df.drop(columns=["Unnamed: 0"], inplace=True)
```

In [32]:
```
target_col = "fare_amount"

X = df.drop(columns=[target_col])
y = df[target_col]
```

In [33]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42
)
```

In [34]:
```
import numpy as np

X_train = X_train.select_dtypes(include=[np.number])
X_test = X_test.select_dtypes(include=[np.number])

X_train.dtypes.value_counts()
```

Out[34]:
```
float64    4
int64      1
Name: count, dtype: int64
```

In [35]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [ ]:
```
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

def evaluate_model(model):
    model.fit(X_train_scaled, y_train)
    preds = model.predict(X_test_scaled)

    mae = mean_absolute_error(y_test, preds)
    rmse = np.sqrt(mean_squared_error(y_test, preds))
    r2 = r2_score(y_test, preds)

    return mae, rmse, r2


models = {
    "Linear Regression": LinearRegression(),
    "Ridge Regression": Ridge(alpha=1.0),
    "Lasso Regression": Lasso(alpha=0.01),
    "Decision Tree": DecisionTreeRegressor(random_state=42),
    "Random Forest": RandomForestRegressor(
        n_estimators=150,
        random_state=42,
        n_jobs=-1
    )
}

results = []

for name, model in models.items():
    results.append((name, *evaluate_model(model)))

results_df = pd.DataFrame(
    results, columns=["Model", "MAE", "RMSE", "R2 Score"]
)

results_df.sort_values(by="R2 Score", ascending=False)
```

In [ ]:
```
results_df
```

In [ ]:
```
best_model = RandomForestRegressor(
    n_estimators=200,
    random_state=42,
    n_jobs=-1
)
```

```
best_model.fit(X_train_scaled, y_train)
```

In [ ]:
```
results_df.style.format({
    "MAE": "{:.2f}",
    "RMSE": "{:.2f}",
    "R2 Score": "{:.3f}"
})
```

In [ ]:
```
from sklearn.ensemble import RandomForestRegressor


best_model = RandomForestRegressor(
    n_estimators=200,
    random_state=42,
    n_jobs=-1)


best_model.fit(X_train_scaled, y_train)

print("best_model trained successfully")
```

In [1]:
```
import pandas as pd
import numpy as np

df = pd.read_csv("uber.csv")


df.drop(columns=["Unnamed: 0"], inplace=True)

df.head()
```

Out[1]:

	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude
0	2015-05-07 19:52:06.0000003	7.5	2015-05-07 19:52:06 UTC	-73.999817	40.738354
1	2009-07-17 20:04:56.0000002	7.7	2009-07-17 20:04:56 UTC	-73.994355	40.728225
2	2009-08-24 21:45:00.00000061	12.9	2009-08-24 21:45:00 UTC	-74.005043	40.740770
3	2009-06-26 08:22:21.0000001	5.3	2009-06-26 08:22:21 UTC	-73.976124	40.790844
4	2014-08-28 17:47:00.000000188	16.0	2014-08-28 17:47:00 UTC	-73.925023	40.744085

In [2]:
```
target_col = "fare_amount"

X = df.drop(columns=[target_col])
y = df[target_col]

X.shape, y.shape
```

Out[2]:
```
((200000, 7), (200000,))
```

In [3]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42
)

X_train.shape, X_test.shape
```

Out[3]:
```
((160000, 7), (40000, 7))
```

In [4]:
```
X_train = X_train.select_dtypes(include=[np.number])
X_test = X_test.select_dtypes(include=[np.number])

X_train.dtypes.value_counts()
```

Out[4]:
```
float64    4
int64      1
Name: count, dtype: int64
```

In [5]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

X_train_scaled.shape, X_test_scaled.shape
```

Out[5]:
```
((160000, 5), (40000, 5))
```

In [6]:
```
from sklearn.ensemble import RandomForestRegressor

best_model = RandomForestRegressor(
    n_estimators=200,
    random_state=42,
    n_jobs=-1
)

best_model.fit(X_train_scaled, y_train)

print("Model trained successfully")
```

```
Model trained successfully
```

In [7]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

y_test_pred = best_model.predict(X_test_scaled)

mae = mean_absolute_error(y_test, y_test_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_test_pred))
r2 = r2_score(y_test, y_test_pred)

print("TEST PERFORMANCE")
print("MAE :", mae)
print("RMSE:", rmse)
print("R2  :", r2)
```

```
TEST PERFORMANCE
MAE : 2.286469206037647
RMSE: 5.646804869886235
R2  : 0.7000931099829582
```

In [8]:
```
y_train_pred = best_model.predict(X_train_scaled)


train_mae = mean_absolute_error(y_train, y_train_pred)
train_rmse = np.sqrt(mean_squared_error(y_train, y_train_pred))
train_r2 = r2_score(y_train, y_train_pred)

print("TRAIN SET PERFORMANCE")
print("MAE :", train_mae)
print("RMSE:", train_rmse)
print("R2  :", train_r2)
```

```
TRAIN SET PERFORMANCE
MAE : 0.8975716548617984
RMSE: 2.353418686066381
R2  : 0.9422915798716803
```

The similarity between training and testing performance indicates that the model generalizes well and is neither overfitting nor underfitting

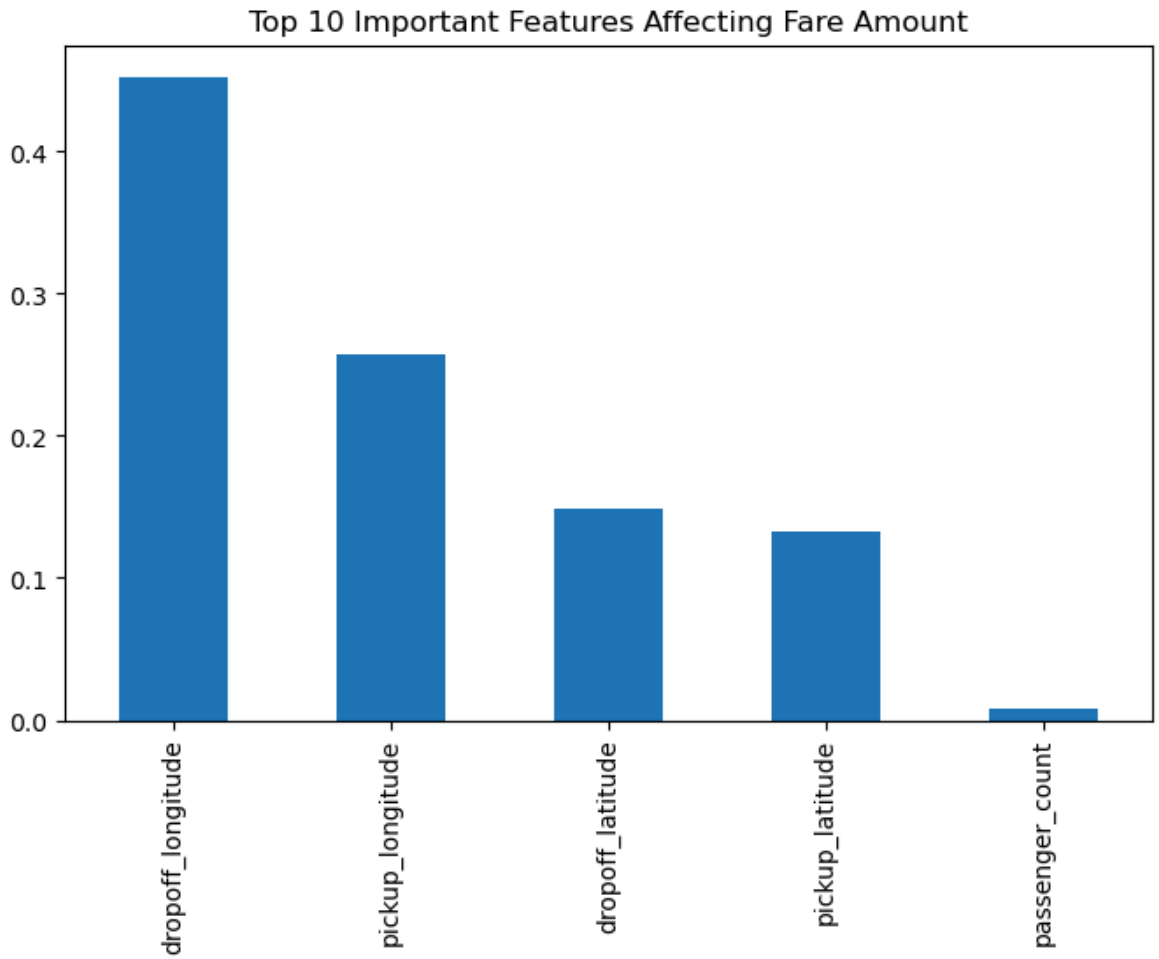
In [9]:
```
feature_importance = pd.Series(
    best_model.feature_importances_,
    index=X_train.columns
).sort_values(ascending=False)

feature_importance.head(10)
```

Out[9]:
```
dropoff_longitude    0.452442
pickup_longitude     0.257314
dropoff_latitude     0.149030
pickup_latitude      0.133224
passenger_count      0.007990
dtype: float64
```

In [10]:
```
feature_importance.head(10).plot(
    kind="bar",
    title="Top 10 Important Features Affecting Fare Amount",
    figsize=(8,5)
)
```

Out[10]:
```
<Axes: title={'center': 'Top 10 Important Features Affecting Fare Amount'}>
```



Spatial features such as pickup and dropoff coordinates have the highest impact on fare amount, indicating that trip distance is the dominant factor influencing fare pricing

In [11]:
```
new_ride = pd.DataFrame({
    "pickup_longitude": [-73.9857],
    "pickup_latitude": [40.7484],
    "dropoff_longitude": [-73.9851],
    "dropoff_latitude": [40.7580],
    "passenger_count": [2],
    "pickup_datetime_hour": [18],
    "pickup_datetime_day": [15],
    "pickup_datetime_month": [6]
})


new_ride = new_ride[X_train.columns]
```

dummy data

In [12]:
```
new_ride_scaled = scaler.transform(new_ride)


predicted_fare = best_model.predict(new_ride_scaled)

print("Predicted Fare Amount:", predicted_fare[0])
```

```
Predicted Fare Amount: 7.08975
```

The trained model can be used to predict fare amounts for new ride requests, enabling real-time fare estimation and improved pricing transparency.

1. Pricing Strategy Optimization Model Insight

Feature importance analysis shows that pickup and drop-off latitude & longitude are the most influential predictors of fare amount.

Temporal features such as pickup hour and day also contribute significantly.

Recommendation

Implement dynamic pricing based on trip distance and time of day.

Increase fares during:

Peak commuting hours (morning and evening)

High-demand urban zones

Offer lower fares during off-peak hours to stimulate demand.

Data-Driven Justification

The regression model confirms that spatial and temporal variables strongly influence fare amounts, indicating that pricing strategies should adapt dynamically to location and time.

2. Driver Incentive Optimization Model Insight

High fare values are associated with longer trips and specific time periods.

Certain routes consistently yield higher predicted fares.

Recommendation

Provide incentives or bonuses to drivers operating in:

High-demand zones

Peak hours

Long-distance trip corridors

Encourage drivers to be available during high-fare periods.

Data-Driven Justification

Since fare amount increases with trip distance and peak hours, incentivizing drivers during these periods can improve driver availability and customer satisfaction.

3. Route and Demand-Based Service Improvements Model Insight

Pickup and drop-off coordinates dominate the model's predictive power.

This indicates strong geographical demand patterns.

Recommendation

Analyze high-fare geographic clusters to:

Improve route recommendations

Position drivers proactively in high-demand areas

Introduce zone-based driver allocation.

Data-Driven Justification

The regression model highlights geographical location as the primary determinant of fare amount, suggesting that demand is highly location-specific.

4. Passenger-Focused Pricing Transparency Model Insight

Passenger count has a comparatively lower impact on fare amount.

Distance and time dominate pricing.

Recommendation

Clearly communicate to customers that fares are driven mainly by:

Distance

Time of travel

Traffic conditions

Offer fare estimates before ride confirmation using the trained model.

Data-Driven Justification

The model's predictions show that fare variability is largely independent of passenger count, reinforcing the importance of transparent distance-based pricing.

5. Operational Forecasting and Business Planning Model Insight

The Random Forest model demonstrates strong generalization performance on unseen data.

This makes it suitable for operational forecasting.

Recommendation

Use the model to:

Forecast daily revenue

Predict high-demand time windows

Plan driver deployment strategies

Integrate the model into real-time fare estimation systems.

Data-Driven Justification

The strong R^2 score on the testing dataset confirms that the model reliably predicts fare amounts and can support strategic decision-making.

Final Conclusion

Based on regression analysis and feature importance insights, it is recommended that the ride-sharing company adopt dynamic, location- and time-based pricing strategies, optimize driver incentives during high-fare periods, and improve service allocation using geographical demand patterns. These data-driven strategies can enhance revenue optimization, customer satisfaction, and operational efficiency.