Predicting Customer Bookings Using Machine Learning

Project: Boston Work (British Airways)

Analyst: Mridul Vatsal

(NOTE):[Data set used here in notebook is already cleaned using Excel for simplification in Analysis.]

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from imblearn.over_sampling import SMOTE
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
```

In [2]:
```
df = pd.read_csv("travel_booking_data.csv")
print("Dataset Shape:", df.shape)
df.head()
```

```
Dataset Shape: (10000, 7)
```

Out[2]:

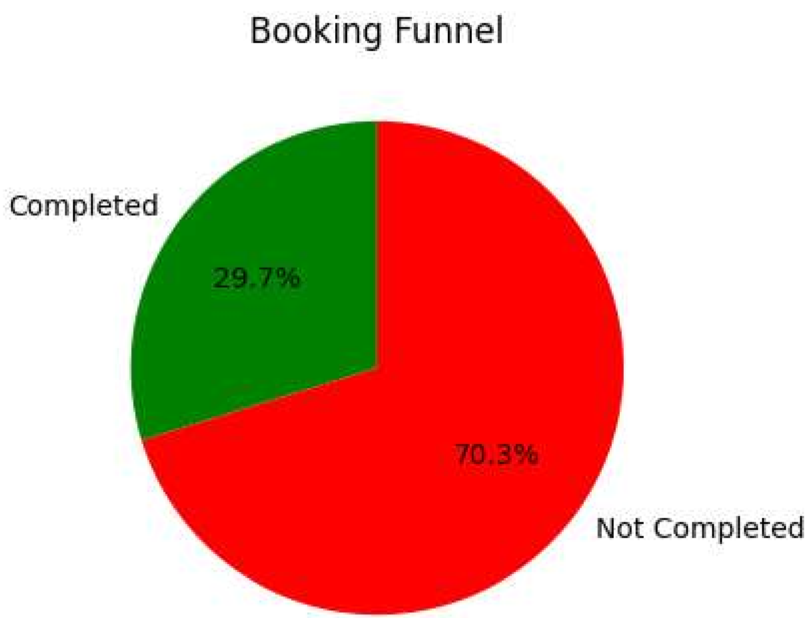
	customer_id	age	gender	trip_type	destination	past_bookings	booking_complete
0	1	56	Male	RoundTrip	Tokyo	2	0
1	2	69	Female	RoundTrip	Paris	3	0
2	3	46	Female	OneWay	New York	1	0
3	4	32	Female	RoundTrip	Tokyo	0	0
4	5	60	Male	RoundTrip	Paris	3	0

In [3]:

```
df["is_frequent_traveler"] = df["past_bookings"] > 2
df["age_group"] = pd.cut(df["age"], bins=[0, 25, 40, 60, 100], labels=["<25", "25-40", "41-60", "60+"])
trip_popularity = df.groupby("destination")["booking_complete"].mean()
df["trip_popularity_score"] = df["destination"].map(trip_popularity)
```

In [4]:

```
total = len(df)
completed = df["booking_complete"].sum()
dropoff = total - completed
labels = ["Completed", "Not Completed"]
values = [completed, dropoff]
plt.figure(figsize=(6,4))
plt.pie(values, labels=labels, autopct='%1.1f%%', startangle=90, colors=["green", "red"])
plt.title("Booking Funnel")
plt.show()
```



In [5]:
```
df = df.drop("customer_id", axis=1)
df_encoded = pd.get_dummies(df, drop_first=True)
X = df_encoded.drop("booking_complete", axis=1)
y = df_encoded["booking_complete"]
```

In [6]:
```
# Normalize for SVM
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# SMOTE for class balancing
smote = SMOTE(random_state=42)
X_res, y_res = smote.fit_resample(X_scaled, y)
```

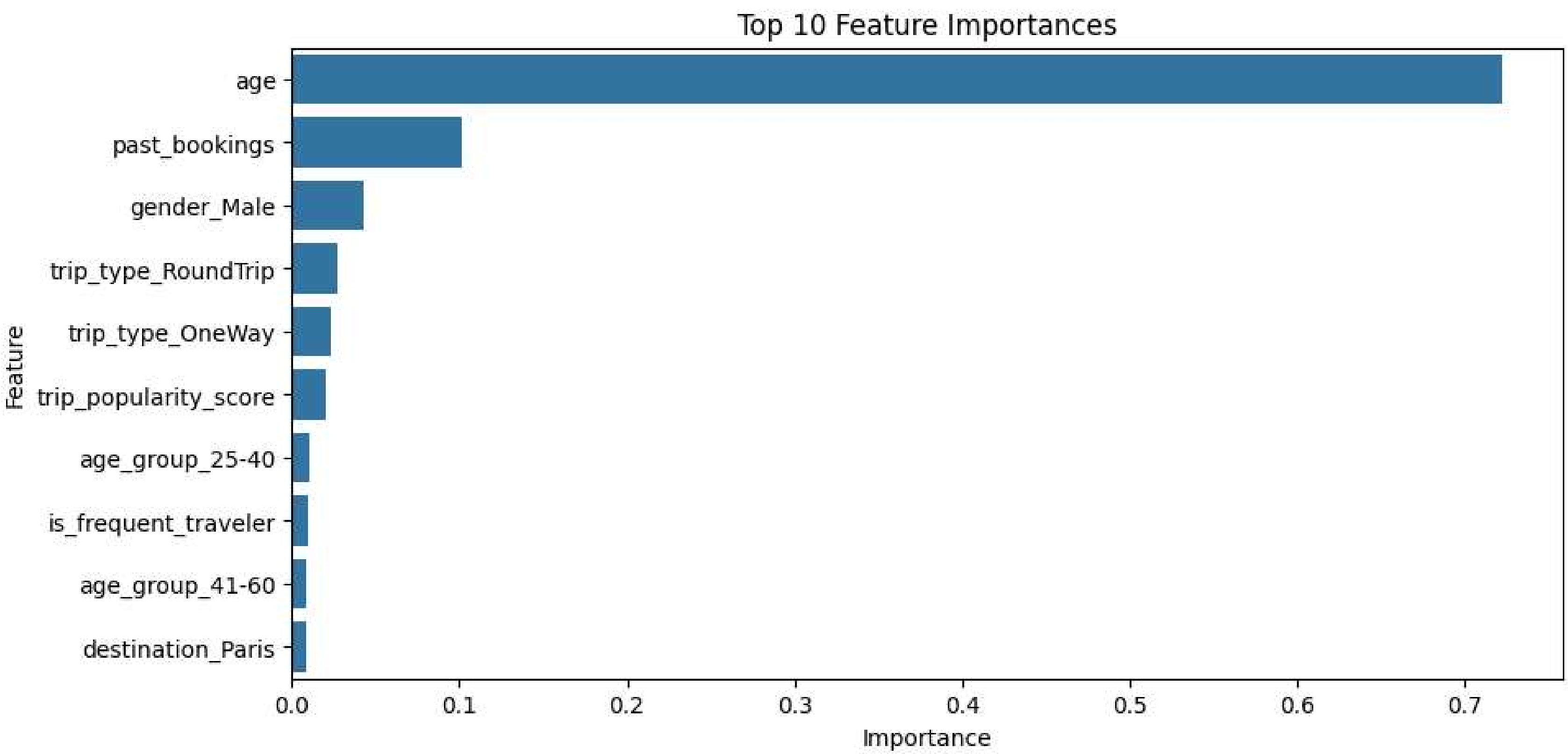
Model Comparison

In [7]:
```
models = {
    "Logistic Regression": LogisticRegression(max_iter=500),
    "Random Forest": RandomForestClassifier(random_state=42),
    "Gradient Boosting": GradientBoostingClassifier(),
    "SVM": SVC(probability=True)
}

results = []
for name, model in models.items():
    X_train, X_test, y_train, y_test = train_test_split(X_res, y_res, test_size=0.2, random_state=42)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    results.append({
        "Model": name,
        "Accuracy": round(accuracy_score(y_test, y_pred), 2),
        "Precision": round(precision_score(y_test, y_pred), 2),
        "Recall": round(recall_score(y_test, y_pred), 2),
        "F1": round(f1_score(y_test, y_pred), 2)
    })

results_df = pd.DataFrame(results)
print("\nModel Comparison Table:")
print(results_df)
```

```
Model Comparison Table:
                 Model  Accuracy  Precision  Recall    F1
0  Logistic Regression      0.51       0.49    0.61  0.54
1        Random Forest      0.64       0.62    0.62  0.62
2    Gradient Boosting      0.68       0.70    0.56  0.62
3                  SVM      0.53       0.50    0.59  0.54
```

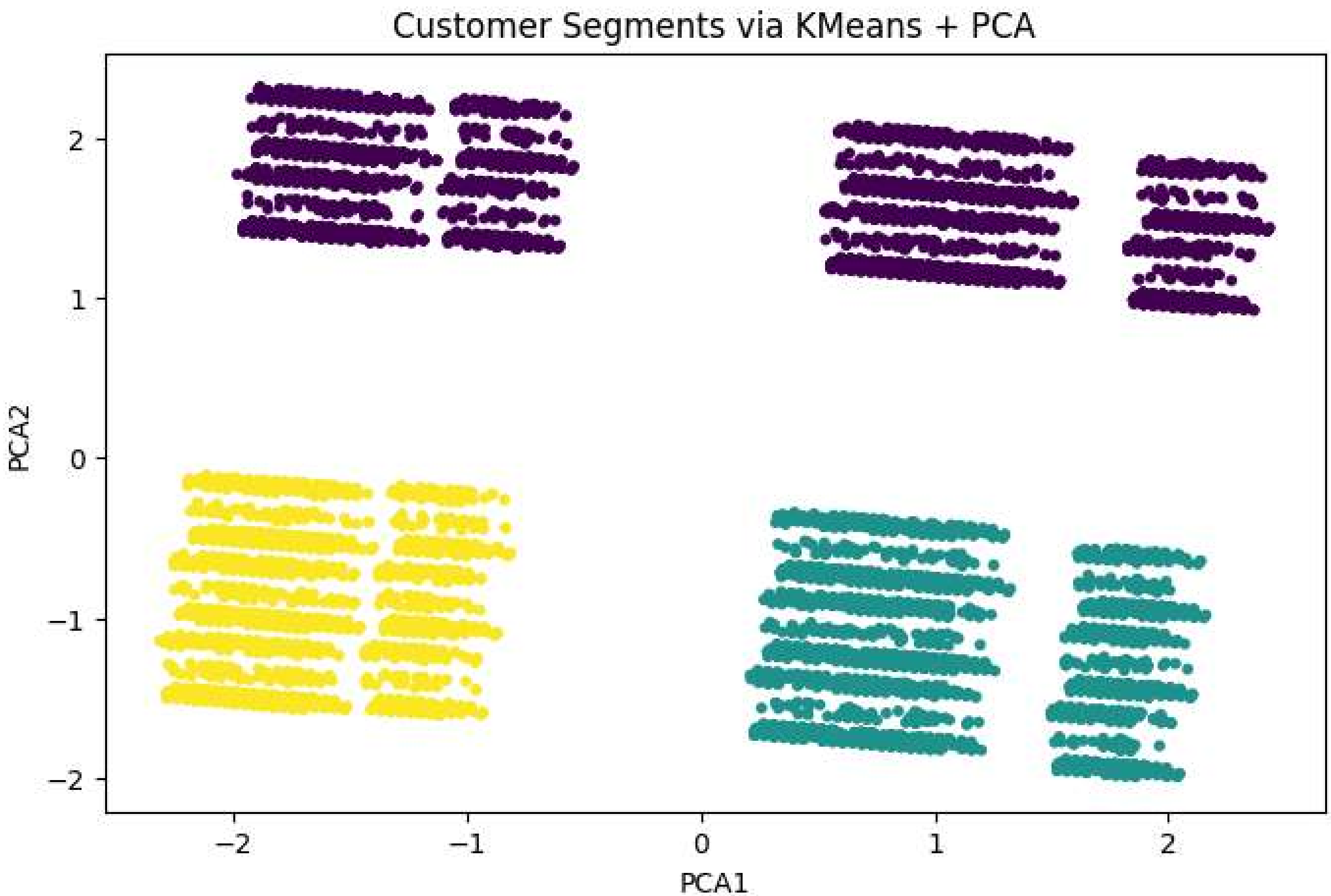
Feature Importance (Random Forest)

In [8]:
```
rf = RandomForestClassifier(random_state=42)
rf.fit(X_train, y_train)
importances = rf.feature_importances_
features = X.columns
imp_df = pd.DataFrame({"Feature": features, "Importance": importances}).sort_values(by="Importance", ascending=False)

plt.figure(figsize=(10,5))
sns.barplot(x="Importance", y="Feature", data=imp_df)
plt.title("Top 10 Feature Importances")
plt.show()
```



Customer Segmentation (KMeans)

In [9]:
```
pca = PCA(n_components=2)
X_seg = pca.fit_transform(X_scaled)
kmeans = KMeans(n_clusters=3, random_state=42)
segments = kmeans.fit_predict(X_seg)

plt.figure(figsize=(8,5))
plt.scatter(X_seg[:, 0], X_seg[:, 1], c=segments, cmap="viridis", s=10)
plt.title("Customer Segments via KMeans + PCA")
plt.xlabel("PCA1")
plt.ylabel("PCA2")
plt.show()
```



Predict Function

In [10]:
```
def predict_booking(input_dict):
    df_input = pd.DataFrame([input_dict])
```

```
    df_input["is_frequent_traveler"] = df_input["past_bookings"] > 2
    df_input["age_group"] = pd.cut(df_input["age"], bins=[0, 25, 40, 60, 100], labels=["<25", "25-40", "41-60", "60+'
    df_input["trip_popularity_score"] = trip_popularity.get(df_input["destination"].values[0], 0.5)
    df_input_encoded = pd.get_dummies(df_input)
    df_input_encoded = df_input_encoded.reindex(columns=X.columns, fill_value=0)
    scaled_input = scaler.transform(df_input_encoded)
    pred = rf.predict(scaled_input)[0]
    prob = rf.predict_proba(scaled_input)[0][1]
    return {"Prediction": int(pred), "Probability": round(prob, 2)}
```

Insights & Roadmap

- RoundTrips and past booking history are key predictors.

- Destinations like Tokyo and Paris have higher booking completion rates.

- Customer clusters show different behavior patterns.

- Future Work:

* Hyperparameter tuning (GridSearchCV)

* Real-time prediction interface (Streamlit/Flask)

* Booking time patterns over months