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

In [2]:
```
from google.colab import files
uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving ncr_ride_bookings.csv to ncr_ride_bookings.csv
```

In [3]:
```
df = pd.read_csv("ncr_ride_bookings.csv")
df.head(20)
```

Out[3]:

	Date	Time	Booking ID	Booking Status	Customer ID	Vehicle Type	Pickup Location	Dr Locati
0	2024-03-23	12:29:38	"CNR5884300"	No Driver Found	"CID1982111"	eBike	Palam Vihar	Jhil
1	2024-11-29	18:01:39	"CNR1326809"	Incomplete	"CID4604802"	Go Sedan	Shastri Nagar	Gurga Sector
2	2024-08-23	08:56:10	"CNR8494506"	Completed	"CID9202816"	Auto	Khandsa	Malv Nag
3	2024-10-21	17:17:25	"CNR8906825"	Completed	"CID2610914"	Premier Sedan	Central Secretariat	Inder
4	2024-09-16	22:08:00	"CNR1950162"	Completed	"CID9933542"	Bike	Ghitorni Village	Kh Marl
5	2024-02-06	09:44:56	"CNR4096693"	Completed	"CID4670564"	Auto	AIIMS	Narsinghp
6	2024-06-17	15:45:58	"CNR2002539"	Completed	"CID6800553"	Go Mini	Vaishali	Punja Ba
7	2024-03-19	17:37:37	"CNR6568000"	Completed	"CID8610436"	Auto	Mayur Vihar	Cyber H
8	2024-09-14	12:49:09	"CNR4510807"	No Driver Found	"CID7873618"	Go Sedan	Noida Sector 62	Noi Sector
9	2024-12-16	19:06:48	"CNR7721892"	Incomplete	"CID5214275"	Auto	Rohini	Ada Nag
10	2024-06-14	16:24:12	"CNR9070334"	Completed	"CID6680340"	Auto	Udyog Bhawan	Dwai Sector
11	2024-09-18	08:09:38	"CNR9551927"	No Driver Found	"CID7568143"	Auto	Vidhan Sabha	AIII
12	2024-06-25	22:44:15	"CNR4386945"	Cancelled by Driver	"CID5543520"	eBike	Patel Chowk	Khe Daula T
13	2024-09-11	19:29:39	"CNR2987763"	Completed	"CID2669710"	Go Mini	Malviya Nagar	Ghito Villa
14	2024-10-18	18:28:53	"CNR8962232"	Completed	"CID1789354"	Go Mini	Madipur	GTB Nag
15	2024-06-07	15:05:35	"CNR2390352"	Completed	"CID5432215"	Auto	Jama Masjid	Kh Marl
16	2024-07-01	10:51:16	"CNR3221338"	Completed	"CID2581698"	Premier Sedan	IGI Airport	Madip

	Date	Time	Booking ID	Booking Status	Customer ID	Vehicle Type	Pickup Location	Dr Locati
17	2024-12-15	15:08:25	"CNR6739317"	Cancelled by Driver	"CID8682675"	Go Sedan	Vinobapuri	GTB Na
18	2024-11-24	09:07:10	"CNR6126048"	Cancelled by Customer	"CID1060329"	eBike	Kashmere Gate	Anand Vi
19	2024-05-24	19:53:57	"CNR9465840"	Cancelled by Driver	"CID9046501"	eBike	Pitampura	Rajiv Na

20 rows × 21 columns

Explore Data Analysis

In [4]:
```
df.columns
```

Out[4]:
```
Index(['Date', 'Time', 'Booking ID', 'Booking Status', 'Customer ID',
       'Vehicle Type', 'Pickup Location', 'Drop Location', 'Avg VTAT',
       'Avg CTAT', 'Cancelled Rides by Customer',
       'Reason for cancelling by Customer', 'Cancelled Rides by Driver',
       'Driver Cancellation Reason', 'Incomplete Rides',
       'Incomplete Rides Reason', 'Booking Value', 'Ride Distance',
       'Driver Ratings', 'Customer Rating', 'Payment Method'],
      dtype='object')
```

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150000 entries, 0 to 149999
Data columns (total 21 columns):
 #   Column                             Non-Null Count   Dtype  
---  ------                             --------------   -----  
 0   Date                               150000 non-null  object 
 1   Time                               150000 non-null  object 
 2   Booking ID                         150000 non-null  object 
 3   Booking Status                     150000 non-null  object 
 4   Customer ID                        150000 non-null  object 
 5   Vehicle Type                       150000 non-null  object 
 6   Pickup Location                    150000 non-null  object 
 7   Drop Location                      150000 non-null  object 
 8   Avg VTAT                           139500 non-null  float64
 9   Avg CTAT                           102000 non-null  float64
 10  Cancelled Rides by Customer        10500 non-null   float64
 11  Reason for cancelling by Customer  10500 non-null   object 
 12  Cancelled Rides by Driver          27000 non-null   float64
 13  Driver Cancellation Reason         27000 non-null   object 
 14  Incomplete Rides                   9000 non-null    float64
 15  Incomplete Rides Reason            9000 non-null    object 
 16  Booking Value                      102000 non-null  float64
 17  Ride Distance                      102000 non-null  float64
 18  Driver Ratings                     93000 non-null   float64
 19  Customer Rating                    93000 non-null   float64
 20  Payment Method                     102000 non-null  object 
dtypes: float64(9), object(12)
memory usage: 24.0+ MB
```

In [6]:
```
df.isna().sum()
```

Out[6]:

	0
Date	0
Time	0
Booking ID	0
Booking Status	0
Customer ID	0
Vehicle Type	0
Pickup Location	0
Drop Location	0
Avg VTAT	10500
Avg CTAT	48000
Cancelled Rides by Customer	139500
Reason for cancelling by Customer	139500
Cancelled Rides by Driver	123000
Driver Cancellation Reason	123000
Incomplete Rides	141000
Incomplete Rides Reason	141000
Booking Value	48000
Ride Distance	48000
Driver Ratings	57000
Customer Rating	57000
Payment Method	48000

dtype: int64

In [7]:

```
#Threshold to drop
threshold = 0.8
df = df.loc[:, df.isnull().mean() < threshold]

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150000 entries, 0 to 149999
Data columns (total 15 columns):
 #   Column           Non-Null Count   Dtype  
---  ------           --------------   -----  
 0   Date             150000 non-null  object 
 1   Time             150000 non-null  object 
 2   Booking ID       150000 non-null  object 
 3   Booking Status   150000 non-null  object 
 4   Customer ID      150000 non-null  object 
 5   Vehicle Type     150000 non-null  object 
 6   Pickup Location  150000 non-null  object 
 7   Drop Location    150000 non-null  object 
 8   Avg VTAT         139500 non-null  float64
 9   Avg CTAT         102000 non-null  float64
 10  Booking Value    102000 non-null  float64
 11  Ride Distance    102000 non-null  float64
 12  Driver Ratings   93000 non-null   float64
 13  Customer Rating  93000 non-null   float64
 14  Payment Method   102000 non-null  object 
dtypes: float64(6), object(9)
memory usage: 17.2+ MB
```

In [8]:

```
#dtypes conversion
#Date and Time
df['Date'] = pd.to_datetime(df['Date'])
df['Time'] = pd.to_datetime(df['Time'])

#BookingID and CustomerID
df['Booking ID'] = df['Booking ID'].astype(str)
df['Customer ID'] = df['Customer ID'].astype(str)


#categorical columns
cats_columns = ["Booking Status", "Vehicle Type", "Pickup Location","Drop Location","Payment Method"]

df[cats_columns] = df[cats_columns].astype("category")
df.info()
```

```
/tmp/ipython-input-2830177588.py:4: UserWarning: Could not infer format, so each element will be parsed individually, falling back to `dateutil`. To ensure parsing is consistent and as-expected, please specify a format.
  df['Time'] = pd.to_datetime(df['Time'])
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150000 entries, 0 to 149999
Data columns (total 15 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   Date              150000 non-null  datetime64[ns]
 1   Time              150000 non-null  datetime64[ns]
 2   Booking ID        150000 non-null  object
 3   Booking Status    150000 non-null  category
 4   Customer ID       150000 non-null  object
 5   Vehicle Type      150000 non-null  category
 6   Pickup Location   150000 non-null  category
 7   Drop Location     150000 non-null  category
 8   Avg VTAT          139500 non-null  float64
 9   Avg CTAT          102000 non-null  float64
 10  Booking Value     102000 non-null  float64
 11  Ride Distance     102000 non-null  float64
 12  Driver Ratings    93000 non-null   float64
 13  Customer Rating   93000 non-null   float64
 14  Payment Method    102000 non-null  category
dtypes: category(5), datetime64[ns](2), float64(6), object(2)
memory usage: 12.5+ MB
```

In [9]:

```
#drop "" in values
df["Booking ID"] = df["Booking ID"].str.replace('"', '', regex=False)
df["Customer ID"] = df["Customer ID"].str.replace('"', '', regex=False)
```

In [10]:

```
print(df["Booking Status"].value_counts())
print(df[["Vehicle Type", "Avg VTAT", "Avg CTAT"]].isna().sum())
```

```
Booking Status
Completed                93000
Cancelled by Driver      27000
Cancelled by Customer    10500
No Driver Found          10500
Incomplete                9000
Name: count, dtype: int64
Vehicle Type        0
Avg VTAT        10500
Avg CTAT        48000
dtype: int64
```

In [11]:

```
#fillna()
df["Avg VTAT"] = df["Avg VTAT"].fillna(df["Avg VTAT"].median())
df["Avg CTAT"] = df["Avg CTAT"].fillna(df["Avg CTAT"].median())
```

In [12]:

```
#Find correlation
corr = df.corr(numeric_only=True)
corr
```

Out[12]:

	Avg VTAT	Avg CTAT	Booking Value	Ride Distance	Driver Ratings	Customer Rating
Avg VTAT	1.000000	0.050914	0.002259	0.063005	-0.005439	-0.003945
Avg CTAT	0.050914	1.000000	0.000216	0.101503	0.000807	0.001000
Booking Value	0.002259	0.000216	1.000000	0.005174	-0.000249	-0.000287
Ride Distance	0.063005	0.101503	0.005174	1.000000	-0.001875	0.004514
Driver Ratings	-0.005439	0.000807	-0.000249	-0.001875	1.000000	-0.001010
Customer Rating	-0.003945	0.001000	-0.000287	0.004514	-0.001010	1.000000

In [13]:

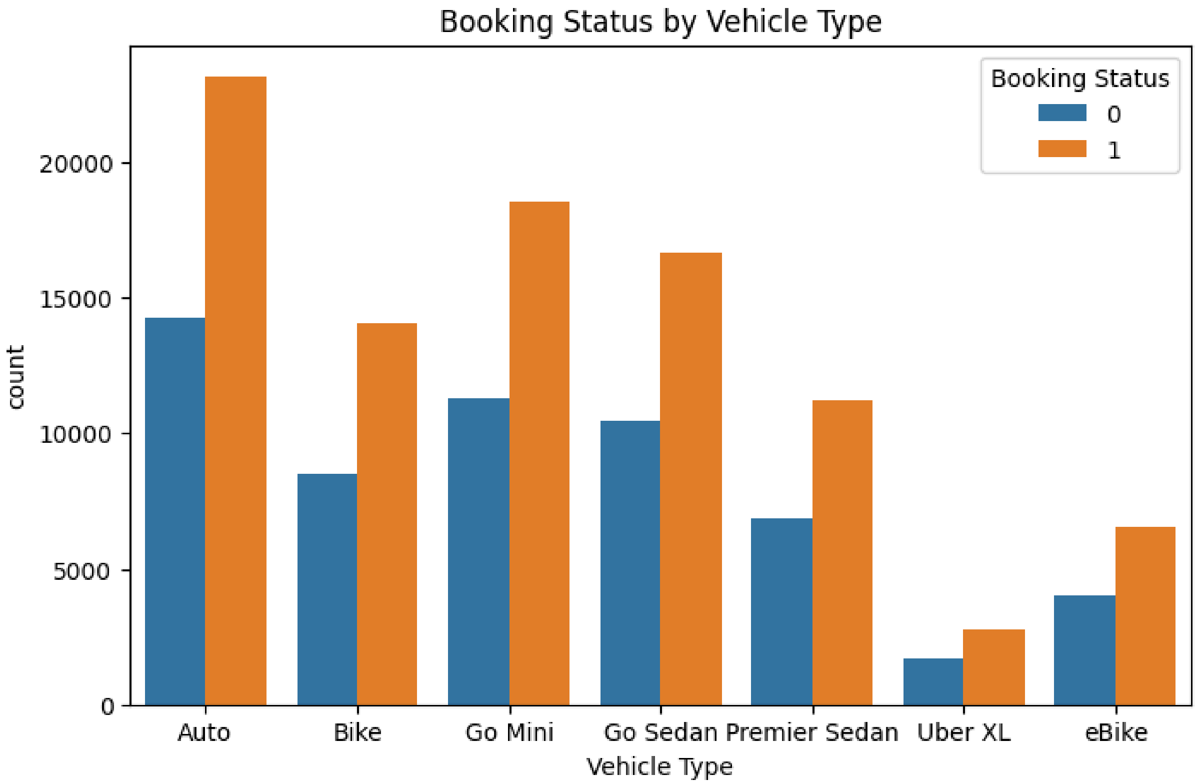
```
#Convert df["Booking Status"]
df["Booking Status"] = df["Booking Status"].apply(lambda x: 1 if x=="Completed" 
else 0)
print(df["Booking Status"].value_counts())
```

```
Booking Status
1    93000
0    57000
Name: count, dtype: int64
```

In [28]:

```
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(7,5))
sns.countplot(x="Booking Status", data=df, order=df["Booking 
Status"].value_counts().index)
plt.title("Distribution of Booking Status")
plt.show()
```

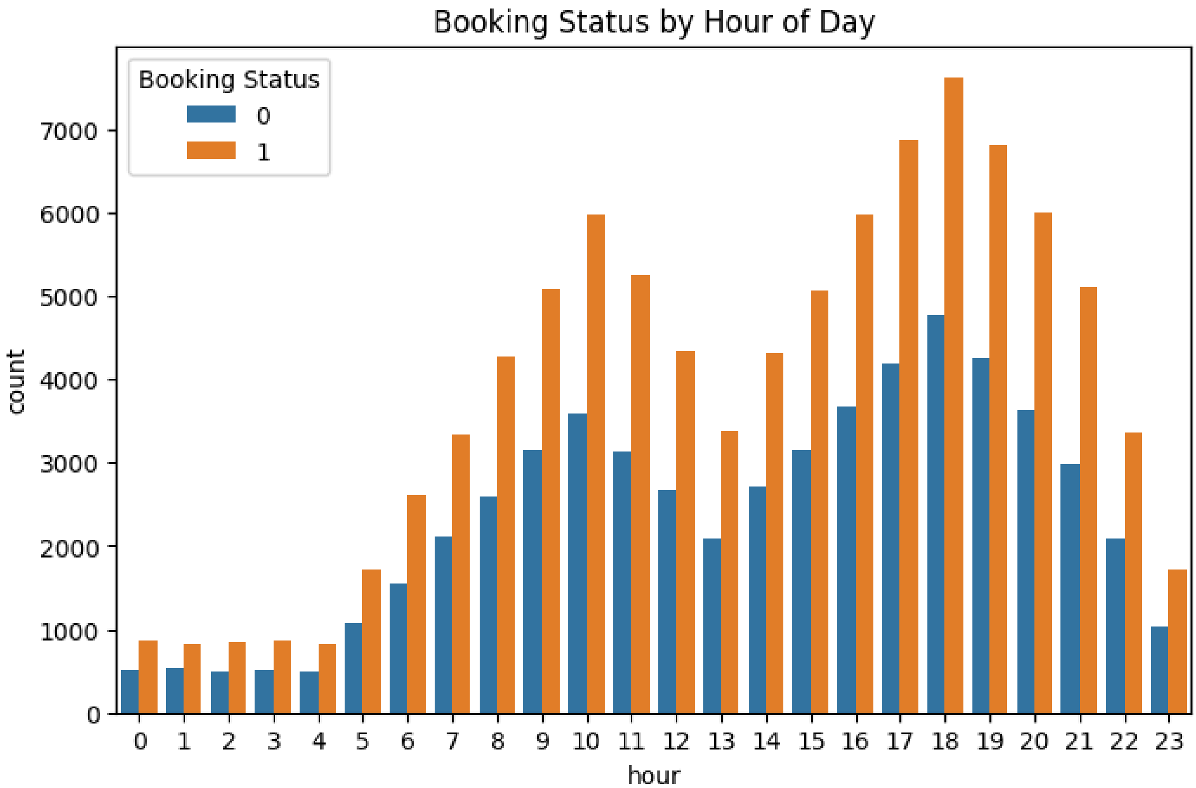
In [30]:
```
#BookingStatus by Vehicle Type
plt.figure(figsize=(8,5))
sns.countplot(x="Vehicle Type", hue="Booking Status", data=df)
plt.title("Booking Status by Vehicle Type")
plt.show()
```



The distribution of booking status across vehicle types shows clear differences in performance. Auto, Go Mini, and Go Sedan rides dominate the dataset and have the highest number of completed rides, though failures are still significant. Bike and eBike trips exhibit proportionally higher failure rates, which may reflect limited availability or rider preferences. Uber XL represents the smallest share of rides overall, but with a relatively balanced split between successful and failed bookings. This suggests that vehicle type is an important predictor of booking outcomes.

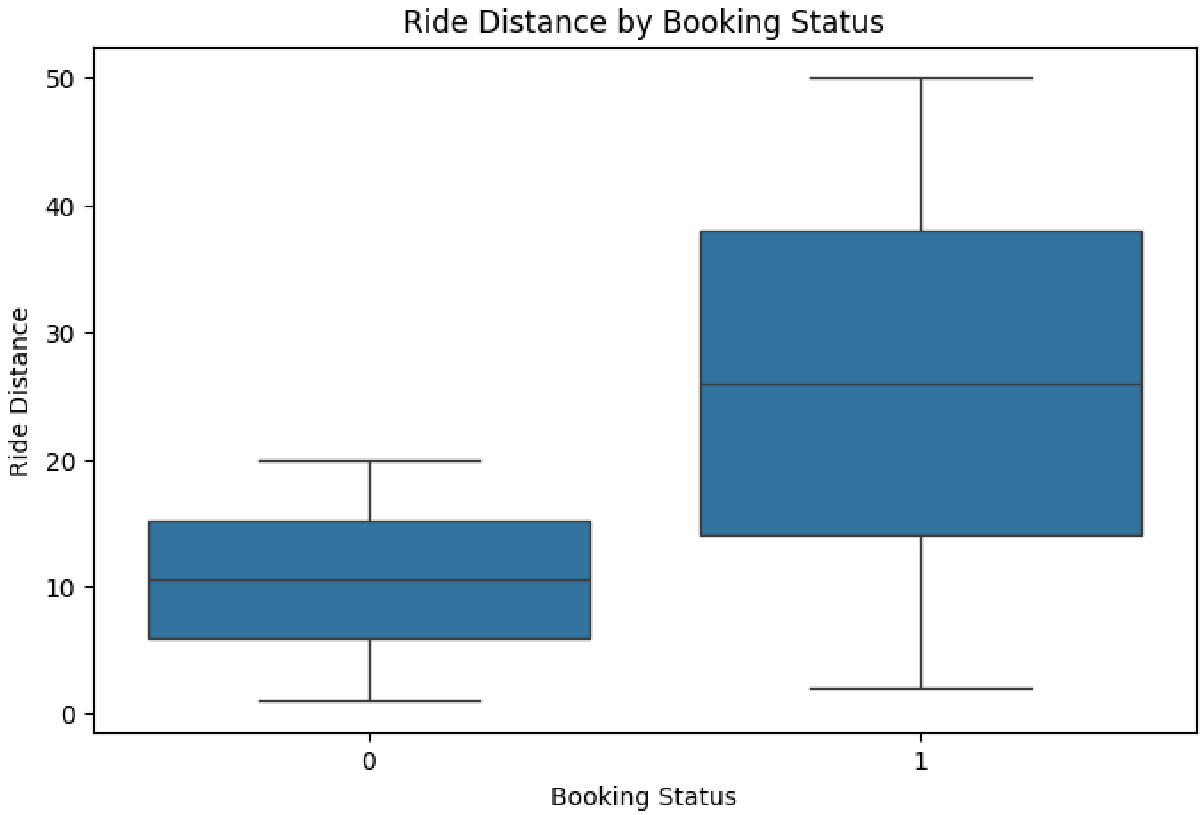
In [33]:

```
#BookingStatus by Hour of Day
df["hour"] = df["Time"].dt.hour
plt.figure(figsize=(8,5))
sns.countplot(x="hour", hue="Booking Status", data=df)
plt.title("Booking Status by Hour of Day")
plt.show()
```



Booking activity increases sharply after 6 AM, peaks during morning commute 8–10 AM, and again during evening rush hours 5–8 PM. Failures rise in parallel with completions, showing that high demand periods lead to more unsuccessful bookings. Midday 11 AM–3 PM sees fewer rides overall but a steadier success rate. The pattern highlights how supply–demand mismatches during peak hours contribute to booking failures, a key operational insight for ride allocation.

In [35]:
```
plt.figure(figsize=(8,5))
sns.boxplot(x="Booking Status", y="Ride Distance", data=df)
plt.title("Ride Distance by Booking Status")
plt.show()
```



The boxplot shows a clear difference in ride distance between successful and failed bookings. Completed rides (1) generally span a wider range, with many medium-to-long trips (up to ~50 km), while failed bookings (0) are concentrated at shorter distances, mostly under 20 km. This suggests that customers attempting short trips may cancel more often, or drivers may deprioritize them due to lower fares. Longer trips, while less frequent, appear more likely to be completed. Overall, ride distance emerges as an informative feature for predicting booking status.

PREDICTION

In [36]:
```
from sklearn.model_selection import train_test_split

# Target
y = df["Booking Status"]

# Features (start small & clean)
X = df[[
    "Vehicle Type",
    "Avg VTAT",
    "Avg CTAT",
    "Ride Distance",
    "hour"
]]

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, stratify=y, random_state=1
)
```

In [37]:
```
from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score

#ColumnTransformer
preprocessor = ColumnTransformer(
    transformers=[
        ("cat", OneHotEncoder(handle_unknown="ignore"), ["Vehicle Type"]),
        ("num", StandardScaler(), ["Avg VTAT", "Avg CTAT"])
    ]
)

#Pipeline
pipelines = [
    ("Decision Tree", Pipeline([
        ("prep", preprocessor),
        ("clf", DecisionTreeClassifier(max_depth=5, random_state=1))
    ])),
    ("Random Forest", Pipeline([
        ("prep", preprocessor),
        ("clf", RandomForestClassifier(n_estimators=200, random_state=1,
n_jobs=-1))
    ])),
    ("XGBoost", Pipeline([
        ("prep", preprocessor),
        ("clf", XGBClassifier(
            n_estimators=300, learning_rate=0.1, max_depth=6,
            subsample=0.8, colsample_bytree=0.8, random_state=1,
            eval_metric="logloss", n_jobs=-1
        ))
    ])),
]

#Train & evaluate
for name, pipe in pipelines:
    pipe.fit(X_train, y_train)
    y_pred = pipe.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    print(f"{name} Accuracy: {acc:.4f}")
```

```
Decision Tree Accuracy: 0.9529
Random Forest Accuracy: 0.9346
XGBoost Accuracy: 0.9527
```

OPTIMIZATION

In [38]:
```
from sklearn.model_selection import GridSearchCV
```

```
# Decision Tree
param_dt = {
    "clf__max_depth": [5, 10],
    "clf__min_samples_split": [2, 5]
}

# Random Forest
param_rf = {
    "clf__n_estimators": [100, 200],
    "clf__max_depth": [10, None],
    "clf__min_samples_split": [2, 5]
}

# XGBoost
param_xgb = {
    "clf__n_estimators": [200, 400],
    "clf__max_depth": [4, 6],
    "clf__learning_rate": [0.05, 0.1],
    "clf__subsample": [0.8, 1.0]
}
```

In [39]:

```
grid_rf = GridSearchCV(
    estimator=pipelines[1][1],
    param_grid=param_rf,
    cv=3,
    scoring="accuracy",
    n_jobs=-1,
    verbose=2
)

grid_rf.fit(X_train, y_train)

print("Best RF params:", grid_rf.best_params_)
print("Best RF score (CV):", grid_rf.best_score_)
```

```
Fitting 3 folds for each of 8 candidates, totalling 24 fits
Best RF params: {'clf__max_depth': 10, 'clf__min_samples_split': 5,
'clf__n_estimators': 100}
Best RF score (CV): 0.9531499999999999
```

EVALUATION

In [40]:

```
from sklearn.model_selection import cross_val_score

# Run 5-fold cross-validation for each pipeline
for name, pipe in pipelines:
    scores = cross_val_score(pipe, X, y, cv=5, scoring="accuracy", n_jobs=-1)
    print(f"{name}: mean={scores.mean():.4f}, std={scores.std():.4f}")
```

```
Decision Tree: mean=0.9531, std=0.0010
/usr/local/lib/python3.12/dist-
packages/joblib/externals/loky/process_executor.py:782: UserWarning: A worker
stopped while some jobs were given to the executor. This can be caused by a too
short worker timeout or by a memory leak.
  warnings.warn(
Random Forest: mean=0.9370, std=0.0006
```

```
/usr/local/lib/python3.12/dist-
packages/joblib/externals/loky/process_executor.py:782: UserWarning: A worker
stopped while some jobs were given to the executor. This can be caused by a too
short worker timeout or by a memory leak.
  warnings.warn(
XGBoost: mean=0.9530, std=0.0010
```

In [41]:

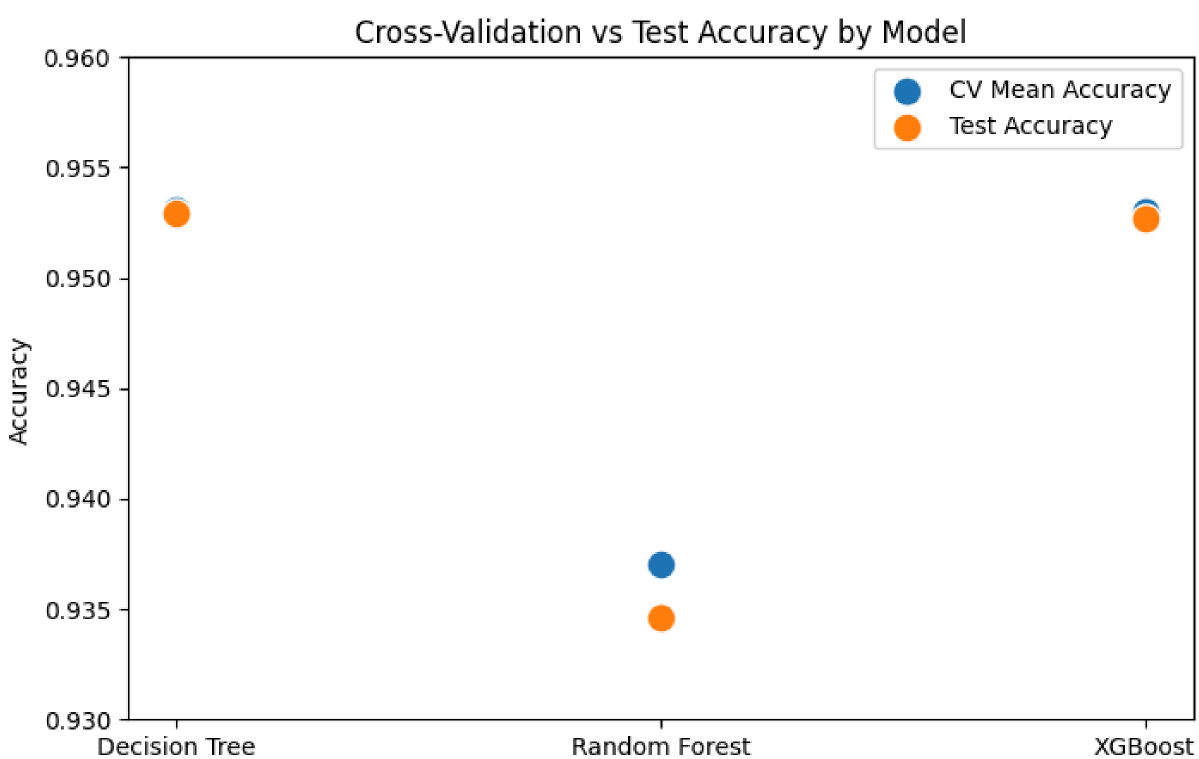
```python
from sklearn.metrics import confusion_matrix, classification_report

# Loop through the trained pipelines and print confusion matrix and classification
report
for name, pipe in pipelines:
    y_pred = pipe.predict(X_test)

    print(f"\n{name} - Confusion Matrix:")
    print(confusion_matrix(y_test, y_pred))

    print(f"\n{name} - Classification Report:")
    print(classification_report(y_test, y_pred, target_names=["Not Booked",
"Booked"]))
```

```
Decision Tree - Confusion Matrix:
[[10046  1354]
 [   59 18541]]

Decision Tree - Classification Report:
              precision    recall  f1-score   support

  Not Booked       0.99      0.88      0.93     11400
      Booked       0.93      1.00      0.96     18600

    accuracy                           0.95     30000
   macro avg       0.96      0.94      0.95     30000
weighted avg       0.96      0.95      0.95     30000


Random Forest - Confusion Matrix:
[[10207  1193]
 [  768 17832]]

Random Forest - Classification Report:
              precision    recall  f1-score   support

  Not Booked       0.93      0.90      0.91     11400
      Booked       0.94      0.96      0.95     18600

    accuracy                           0.93     30000
   macro avg       0.93      0.93      0.93     30000
weighted avg       0.93      0.93      0.93     30000


XGBoost - Confusion Matrix:
[[10049  1351]
 [   68 18532]]

XGBoost - Classification Report:
              precision    recall  f1-score   support

  Not Booked       0.99      0.88      0.93     11400
      Booked       0.93      1.00      0.96     18600

    accuracy                           0.95     30000
   macro avg       0.96      0.94      0.95     30000
weighted avg       0.96      0.95      0.95     30000
```

The evaluation of the models on the test set shows that both Decision Tree and XGBoost achieved the highest accuracy (95%) with near-perfect recall for bookings, meaning they almost never miss customers who actually book. However, this comes with slightly lower precision, as they occasionally predict a booking that does not occur. The Random Forest model performed slightly worse overall (93% accuracy) but offered a more balanced trade-off between precision and recall. This means that prioritizing recall is more valuable since missing a true booking is costlier than predicting one that does not occur, making XGBoost or Decision Tree the preferred models.

VISUALIZATION

In [42]:
```
import seaborn as sns
import matplotlib.pyplot as plt

models = ["Decision Tree", "Random Forest", "XGBoost"]
cv_means = [0.9531, 0.9370, 0.9530]
test_accs = [0.9529, 0.9346, 0.9527]

plt.figure(figsize=(8,5))

# CV points
sns.scatterplot(x=models, y=cv_means, s=150, label="CV Mean Accuracy")

# Test points
sns.scatterplot(x=models, y=test_accs, s=150, label="Test Accuracy")

plt.title("Cross-Validation vs Test Accuracy by Model")
plt.ylim(0.93, 0.96)
plt.ylabel("Accuracy")
plt.show()
```



The scatterplot shows that Decision Tree and XGBoost achieved nearly identical performance, with accuracies around 95.3% on both cross-validation and the test set. Random Forest performed slightly lower, at around 93.5%, but still showed stable results. The close alignment between cross-validation and test accuracies across all models suggests strong generalization without overfitting. Overall, XGBoost is the most reliable choice due to its robustness, while the Decision Tree offers similar accuracy with greater simplicity.

REPORT

A 95% accuracy means the model mostly correctly predicts the booking outcome (booked vs. not booked) for 95 out of 100 customers.

This level of accuracy suggests the model has learned strong patterns from the data and can be trusted for operational use, such as targeting customers at risk of not booking.

However, in practice, accuracy alone isn't enough, you'd also want to check precision, recall, and confusion matrix to understand whether the model is better at catching bookers or non-bookers.

In [25]:

```
from IPython.core.display import display, HTML

display(HTML('''
<style>
  pre {
    white-space: pre-wrap !important;  /* Wrap text */
    word-break: break-word !important; /* Prevent long words from overflowing */
  }
  .output_result {
    max-height: none !important; /* Remove scroll */
  }
  .output_area {
    overflow-y: visible !important; /* Prevent vertical scrolling */
  }
</style>
'''))
```

In [ ]:

```
# @markdown Run this cell to download this notebook as a webpage,
`_NOTEBOOK.html`.

import google, json, nbformat

# Get the current notebook and write it to _NOTEBOOK.ipynb
raw_notebook = google.colab._message.blocking_request("get_ipynb",
                                                      timeout_sec=30)["ipynb"]
with open("_NOTEBOOK.ipynb", "w", encoding="utf-8") as ipynb_file:
  ipynb_file.write(json.dumps(raw_notebook))

# Use nbconvert to convert .ipynb to .html.
!jupyter nbconvert --to html --log-level WARN _NOTEBOOK.ipynb

# Download the .html file.
google.colab.files.download("_NOTEBOOK.html")
```