Introduction

This project aims to develop a predictive model to determine the likelihood of ride cancellations by drivers for San Francisco's Shared Autonomous Rides (SAR) service. By analyzing historical ride data, we seek to identify the key factors influencing cancellation decisions. The outcome of this project will enable SAR management to proactively address potential cancellations, improve service reliability, and enhance overall customer satisfaction. The approach involves a comprehensive data analysis process, including data exploration, preprocessing, feature engineering, model selection, training, and evaluation.

Business goals

Reduce Ride Cancellations → Fewer cancellations = Better customer trust and lower operational losses. Improve Customer Satisfaction → On-time pickups and smooth rides lead to repeat customers and good reviews. Optimize Fleet & Driver Allocation → Use vehicles and drivers more efficiently, based on demand patterns. Boost Operational Efficiency → Reduce idle time, avoid unnecessary bookings, and streamline logistics. Make Data-Driven Decisions → Use insights to guide marketing, pricing, and area expansion strategies.

Analytics goals

Predict Ride Cancellations in Advance → Use ML models to identify bookings with high cancellation risk. Understand Customer Behavior → Analyze booking channels (mobile site, online), preferred areas, time slots, etc. Clean & Prepare Real-World Data → Handle missing values, encode categories, and scale features for better model performance. Compare ML Models for Best Accuracy → Test multiple models (Logistic Regression, Random Forest, SVM, etc.) and pick the most reliable one. Provide Actionable Insights → Visualizations and reports that SAR can actually use to take business actions.

Dataset

The dataset ("SAR Rental.csv") is a random subset of 10,000 SAR total records from 2013. The dataset has the following 19 attributes: 1. Row ID: This is the unique identifier of this dataset. It is a number identifying each record. 2. User ID: This is the identifier of each client. There are many duplicates in this subset, meaning there are many clients who have called multiple rides. 3. Vehicle Model ID: This is an ID that represents the type of vehicle driven for each ride. This also identifies the driver of the vehicle. 4. Travel Type ID: This is an ID that represents the type of travel (1= long distance, 2= point to point, 3= hourly rental). 5. Package ID: This is an ID that represents the type of travel package, with the following descriptions: 1=4hrs & 40kms, 2=8hrs & 80kms, 3=6hrs & 60kms, 4= 10hrs & 100kms, 5=5hrs & 50kms, 6=3hrs & 30kms, 7=12hrs & 120kms. 6. From Area: This is an identifier of the starting area. Available only for point-to-point travel. 7. To Area: This is an identifier of the ending area. Available only for point-to-point travel. 8. From City ID: Unique identifier of the starting city (i.e. suburb cities of San Francisco). 9. To City ID: Unique identifier of the ending city. 10. From Date: Date and time of the requested trip start. 11. To Date: Time stamp of trip end. 12. Online Booking: A binary (0,1) variable representing whether the booking was made online or not. 0 represents no, 1 represents yes. 13. Mobile Site Booking: A binary (0,1) variable representing whether booking was made on their mobile site or not. 0 represents no, 1 represents yes. 14. Booking Created: Date and time of booking created. 15. From Lat: The latitude of the start area. 16. From Long: The longitude of the start area. 17. To Lat: The latitude of the end area. 18. To Long: The longitude of the end area. 19. Car Cancellation: The target variable. A binary (0,1) variable representing whether or not the ride was cancelled. 0 means no, 1 means yes.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
# 1. Load the dataset
df = pd.read_csv('SAR Rental.csv')
print(df.head())
```

	row#	user_id	vehicle_model_id	package_id	travel_type_id	from_area_id
0	1	17712	12	NaN	2	1021.0
1	2	17037	12	NaN	2	455.0
2	3	761	12	NaN	2	814.0
3	4	868	12	NaN	2	297.0
4	5	21716	28	NaN	2	1237.0

	to_area_id	from_city_id	to_city_id	from_date	to_date
0	1323.0	NaN	NaN	1/1/2013 22:33	NaN
1	1330.0	NaN	NaN	1/1/2013 12:43	NaN
2	393.0	NaN	NaN	1/2/2013 0:28	1/3/2013 0:00
3	212.0	NaN	NaN	1/1/2013 13:12	NaN
4	330.0	NaN	NaN	1/1/2013 16:33	NaN

	online_booking	mobile_site_booking	booking_created	from_lat	from_long
0	0	0	1/1/2013 8:01	13.028530	77.54625
1	0	0	1/1/2013 9:59	12.999874	77.67812
2	1	0	1/1/2013 12:14	12.908993	77.68890
3	0	0	1/1/2013 12:42	12.997890	77.61488
4	0	0	1/1/2013 15:07	12.926450	77.61206

	to_lat	to_long	Car_Cancellation
0	12.869805	77.653211	0
1	12.953434	77.706510	0
2	13.199560	77.706880	0
3	12.994740	77.607970	0
4	12.858833	77.589127	0

In [3]:
```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 19 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   row#                 10000 non-null  int64  
 1   user_id              10000 non-null  int64  
 2   vehicle_model_id     10000 non-null  int64  
 3   package_id           1752 non-null   float64
 4   travel_type_id       10000 non-null  int64  
 5   from_area_id         9985 non-null   float64
 6   to_area_id           7909 non-null   float64
 7   from_city_id         3706 non-null   float64
 8   to_city_id           339 non-null    float64
 9   from_date            10000 non-null  object 
 10  to_date              5822 non-null   object 
 11  online_booking       10000 non-null  int64  
 12  mobile_site_booking  10000 non-null  int64  
 13  booking_created      10000 non-null  object 
 14  from_lat             9985 non-null   float64
 15  from_long            9985 non-null   float64
 16  to_lat               7909 non-null   float64
 17  to_long              7909 non-null   float64
 18  Car_Cancellation     10000 non-null  int64  
dtypes: float64(9), int64(7), object(3)
memory usage: 1.4+ MB
None
```

In [4]:

```python
#check for missing values
print("Data before processing: ")
print(df.isnull().sum())

#fill missing values
for col in df.columns:
    df[col] = df[col].fillna(df[col].mode()[0])

#check for missing values
print("Data after processing: ")
print(df.isnull().sum())
```

```
Data before processing:
row#                     0
user_id                  0
vehicle_model_id         0
package_id            8248
travel_type_id           0
from_area_id            15
to_area_id            2091
from_city_id          6294
to_city_id            9661
from_date                0
to_date               4178
online_booking           0
mobile_site_booking      0
booking_created          0
from_lat                15
from_long               15
to_lat                2091
to_long               2091
Car_Cancellation         0
dtype: int64
Data after processing:
row#                  0
user_id                0
vehicle_model_id       0
package_id             0
travel_type_id         0
from_area_id           0
to_area_id             0
from_city_id           0
to_city_id             0
from_date              0
to_date                0
online_booking         0
mobile_site_booking    0
booking_created        0
from_lat               0
from_long              0
to_lat                 0
to_long                0
Car_Cancellation       0
dtype: int64
```

In [5]:
```
cat_cols = ['travel_type_id', 'package_id', 'online_booking', 'mobile_site_booking'
print("\nUnique Values in Categorical Columns:")
for col in cat_cols:
    print(f"{col}: {df[col].unique()}")
```

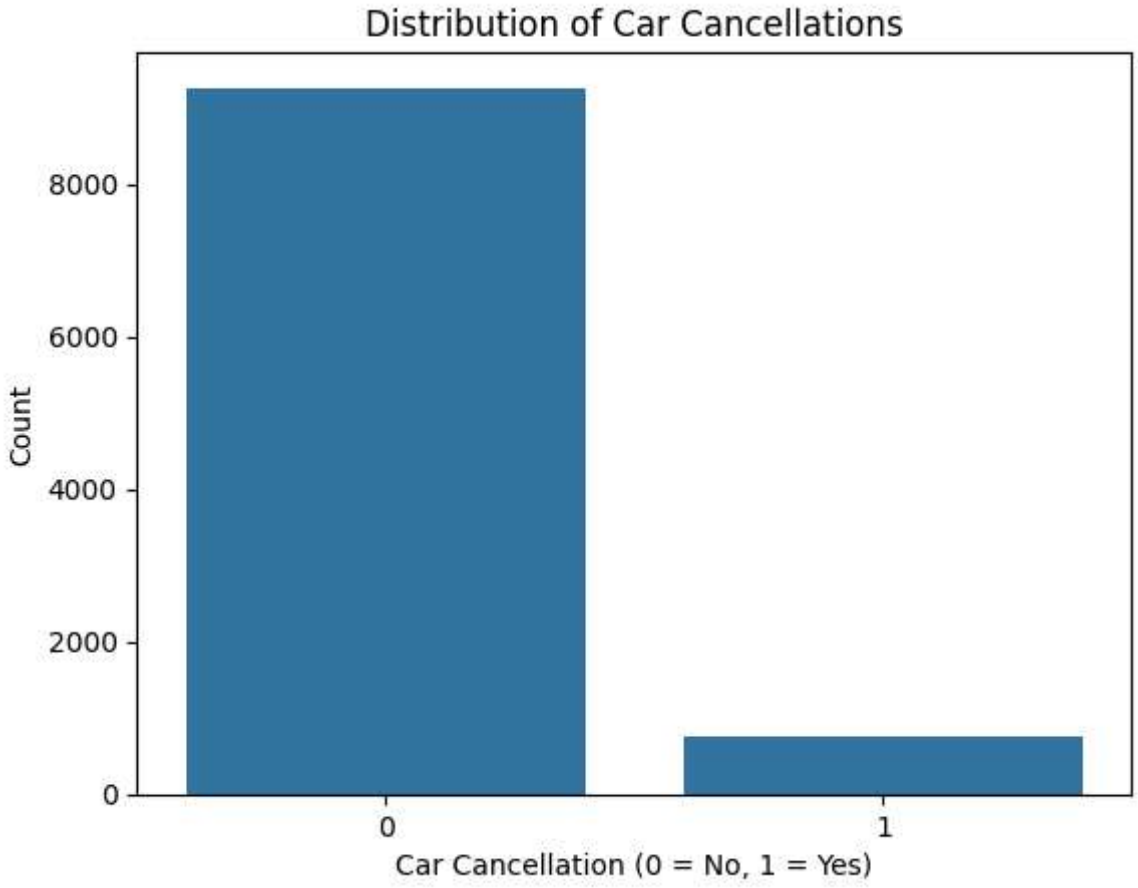
```
Unique Values in Categorical Columns:
travel_type_id: [2 3 1]
package_id: [1. 2. 3. 4. 6. 7. 5.]
online_booking: [0 1]
mobile_site_booking: [0 1]
Car_Cancellation: [0 1]
```

In [6]:
```
# Drop the problematic datetime columns
df=df.drop(columns=['from_date', 'to_date', 'booking_created'])

# Check if the issue is resolved
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 16 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   row#                 10000 non-null  int64  
 1   user_id              10000 non-null  int64  
 2   vehicle_model_id     10000 non-null  int64  
 3   package_id           10000 non-null  float64
 4   travel_type_id       10000 non-null  int64  
 5   from_area_id         10000 non-null  float64
 6   to_area_id           10000 non-null  float64
 7   from_city_id         10000 non-null  float64
 8   to_city_id           10000 non-null  float64
 9   online_booking       10000 non-null  int64  
 10  mobile_site_booking  10000 non-null  int64  
 11  from_lat             10000 non-null  float64
 12  from_long            10000 non-null  float64
 13  to_lat               10000 non-null  float64
 14  to_long              10000 non-null  float64
 15  Car_Cancellation     10000 non-null  int64  
dtypes: float64(9), int64(7)
memory usage: 1.2 MB
None
```

In [7]:
```
# Distribution of cancellations
sns.countplot(data=df, x='Car_Cancellation')
plt.title("Distribution of Car Cancellations")
plt.xlabel("Car Cancellation (0 = No, 1 = Yes)")
plt.ylabel("Count")
plt.show()
```



Note: corr() only works upon the numeric data and not strings. So remember to filter out he columns which have numeric values before applying the dimernsion reduction If you find features with correlation > 0.85 or < -0.85, they may carry similar info — drop one. then dimension reduction is needed.# need of dimension reduction # Select only numeric columns for correlation analysis numeric_df = df.select_dtypes(include=['int64', 'float64']) corr = numeric_df.corr() plt.figure(figsize=(12, 6)) sns.heatmap(corr, annot=True, cmap="coolwarm") plt.title("Feature Correlation Heatmap") plt.show() #steps for reduction from sklearn.decomposition import PCA from sklearn.preprocessing import StandardScaler # Step 1: Extract the two correlated columns geo_data = df[['to_lat', 'to_long']] # Step 2: Standardize the data (important for PCA!) scaler = StandardScaler() geo_scaled = scaler.fit_transform(geo_data) # Step 3: Apply PCA pca = PCA(n_components=1) # reduce to 1 component geo_pca = pca.fit_transform(geo_scaled) # Step 4: Add the new PCA feature back to the original DataFrame df['to_location_pca'] = geo_pca # Step 5: Drop the original columns (optional) df.drop(['to_lat', 'to_long'], axis=1, inplace=True) numeric_df = df.select_dtypes(include=['int64', 'float64']) corr = numeric_df.corr() plt.figure(figsize=(12, 6)) sns.heatmap(corr, annot=True, cmap="coolwarm") plt.title("Feature Correlation Heatmap") plt.show() print(f"Explained Variance by PCA Component: {pca.explained_variance_ratio_[0]*100:.2f}%")

In [8]:
```
from sklearn.preprocessing import LabelEncoder

# Label Encoding categorical columns
le = LabelEncoder()

df['vehicle_model_id'] = le.fit_transform(df['vehicle_model_id'])
df['travel_type_id'] = le.fit_transform(df['travel_type_id'])
df['from_area_id'] = le.fit_transform(df['from_area_id'])
df['to_area_id'] = le.fit_transform(df['to_area_id'])

# Note: 'Car_Cancellation' is already binary, so no need for encoding.
```

In [9]:
```
from sklearn.model_selection import train_test_split

# Assuming 'df' is your cleaned dataframe and preprocessed
X = df.drop(columns=['Car_Cancellation'])  # Features
```

```
y = df['Car_Cancellation']  # Target

# Split dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

In [ ]:

In [10]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Function to evaluate model
def accuracy(model, model_name):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    print(f"\nModel: {model_name}")
    print(f"Accuracy: {accuracy_score(y_test, y_pred):.4f}")
```

In [11]:
```
# Logistic Regression
log_reg = LogisticRegression(solver='liblinear', max_iter=2000)
accuracy(log_reg, "Logistic Regression")
```

```
Model: Logistic Regression
Accuracy: 0.9380
```

Had to change the default solver thats is lbfgs to liblinear bcz lbfgs wasn't able to converge all the data under the max iterations.

In [12]:
```
# Random Forest Classifier
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
accuracy(rf_model, "Random Forest")
```

```
Model: Random Forest
Accuracy: 0.9375
```

In [13]:
```
# K-Nearest Neighbors (KNN)
knn_model = KNeighborsClassifier(n_neighbors=5)
accuracy(knn_model, "K-Nearest Neighbors")
```

```
Model: K-Nearest Neighbors
Accuracy: 0.9275
```

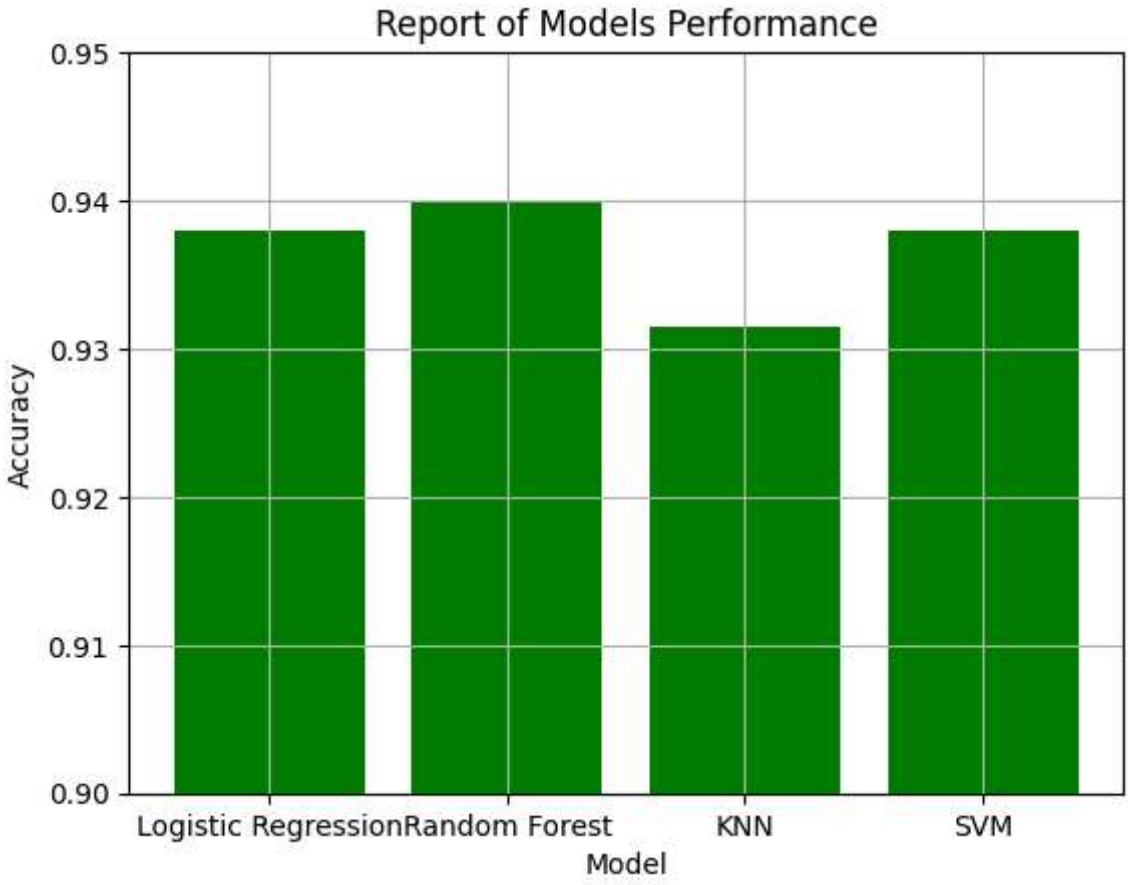
In [14]:
```
# Support Vector Machine (SVM)
svm_model = SVC()
accuracy(svm_model, "Support Vector Machine")
```

```
Model: Support Vector Machine
Accuracy: 0.9380
```

In [15]:
```
model_names = ['Logistic Regression', 'Random Forest', 'KNN', 'SVM']
accuracies = [0.9380, 0.9400, 0.9315, 0.9380]

plt.bar(model_names, accuracies, color='green')
plt.ylabel('Accuracy')
plt.xlabel('Model')
plt.title('Report of Models Performance')
```

```
#adding min and max values on y axis
plt.ylim(0.90, 0.95)
plt.grid(True)
plt.show()
```



The best selected model is 'Random Forest' with accuracy of 94%.

Selected Model: Random Forest

In [16]:
```
# Function to evaluate model
def evaluate(model, model_name):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    print(f"\nModel: {model_name}")
    print(f"Accuracy: {accuracy_score(y_test, y_pred):.4f}")
    print("\nClassification Report:\n", classification_report(y_test, y_pred, zero_


    # Confusion Matrix
    # cm = confusion_matrix(y_test, y_pred)
    # plt.figure(figsize=(6, 6))
    # sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=le.classes_, y
    # plt.title(f'Confusion Matrix - {model_name}')
    # plt.xlabel('Predicted')
    # plt.ylabel('Actual')
    # plt.show()
```

In [17]:
```
# Random Forest Classifier
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
evaluate(rf_model, "Random Forest")
```

```
Model: Random Forest
Accuracy: 0.9375

Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.99      0.97      1876
           1       0.48      0.11      0.18       124

    accuracy                           0.94      2000
   macro avg       0.71      0.55      0.58      2000
weighted avg       0.92      0.94      0.92      2000
```

Conclusion

Random Forest gave the best results — around 94% accuracy. Also, detect some cancelled rides. Logistic Regression and SVM had 93.8% accuracy which is super close to random forest but isnt the best to work with. KNN was okay, but still not great at identifying the rare cancellations

Supporting SAR Goals

SAR wants to: - Improve customer experience - Reduce cancellations - Optimize driver and vehicle assignments We can help by==> 1. We identified patterns — like which areas or time slots have more cancellations. 2. The models can flag future bookings that look risky (more likely to get cancelled), so SAR can take early actions. 3. The analysis gives insights into customer behavior, booking trends, and operational gaps. 4. This helps SAR make smarter, data-based decisions, which saves time, reduces losses, and improves service.

In [ ]:
```
```