# 🚀 SpaceX Falcon 9 First Stage Landing Prediction Analysis

This notebook provides a complete end-to-end data science pipeline for predicting the landing success of SpaceX Falcon 9 first stages. It includes:

- Real data ingestion from SpaceX API
- Data wrangling and exploration
- Visualization
- Machine learning modeling
- Interactive dashboards
- Simple prediction API

Instractor Name: DR.Omar Alnahal

ST name : Alaa Abu Slaeem

St ID : 20211538

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
from datetime import datetime
import requests
from tqdm import tqdm
import warnings
warnings.filterwarnings('ignore')

# Machine Learning
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import (classification_report, confusion_matrix,
                             accuracy_score, precision_score, recall_score, f1_score)

import joblib

plt.style.use('seaborn-v0_8')
sns.set_palette("husl")
```

## 🧪 2. Load Real SpaceX Launch Data from API

We will now load actual Falcon 9 launch data from the official SpaceX public API. This includes flight number, date, launch site, payload, orbit, reuse status, and landing outcome.

```
# Load launch, payload, core and pad data
launches = requests.get('https://api.spacexdata.com/v4/launches').json()
payloads = {p['id']: p for p in requests.get('https://api.spacexdata.com/v4/payloads').json()}
pads = {p['id']: p for p in requests.get('https://api.spacexdata.com/v4/launchpads').json()}
cores = {c['id']: c for c in requests.get('https://api.spacexdata.com/v4/cores').json()}

# Parse launch records
records = []
for launch in tqdm(launches, desc='Processing launches'):
    try:
        flight_number = launch['flight_number']
        date = launch['date_utc']
        success = launch['success']
        site_name = pads[launch['launchpad']]['name'] if launch['launchpad'] in pads else 'Unknown'
        core = launch['cores'][0] if launch['cores'] else {}
```

```
        reused = core.get('reused', False)
        landing_success = core.get('landing_success')
        landing_type = core.get('landing_type', 'Unknown')
        payload_id = launch['payloads'][0] if launch['payloads'] else None
        payload = payloads.get(payload_id, {})
        orbit = payload.get('orbit', 'Unknown')
        mass = payload.get('mass_kg', None)

        records.append({
            'FlightNumber': flight_number,
            'Date': pd.to_datetime(date),
            'LaunchSite': site_name,
            'Orbit': orbit,
            'PayloadMass': mass,
            'Reused': reused,
            'LandingSuccess': landing_success,
            'LandingType': landing_type,
            'MissionSuccess': success
        })
    except Exception as e:
        print(f"Error in {launch['name']}: {e}")

# Create DataFrame
df = pd.DataFrame(records)
df = df.sort_values('FlightNumber').reset_index(drop=True)
df.head()
```

```
Processing launches: 100%|██████████| 205/205 [00:00<00:00, 713.18it/s]
```

| | FlightNumber | Date | LaunchSite | Orbit | PayloadMass | Reused | LandingSuccess | LandingType | MissionSuccess |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2006-03-24 22:30:00+00:00 | Kwajalein Atoll | LEO | 20.0 | False | None | None | False |
| 1 | 2 | 2007-03-21 01:10:00+00:00 | Kwajalein Atoll | LEO | NaN | False | None | None | False |
| 2 | 3 | 2008-08-03 03:34:00+00:00 | Kwajalein Atoll | LEO | NaN | False | None | None | False |
| 3 | 4 | 2008-09-28 23:15:00+00:00 | Kwajalein Atoll | LEO | 165.0 | False | None | None | True |
| 4 | 5 | 2009-07-13 03:35:00+00:00 | Kwajalein Atoll | LEO | 200.0 | False | None | None | True |

Next steps: Generate code with df | View recommended plots | New interactive sheet

## 3. Exploratory Data Analysis (EDA)

In this section, we will:

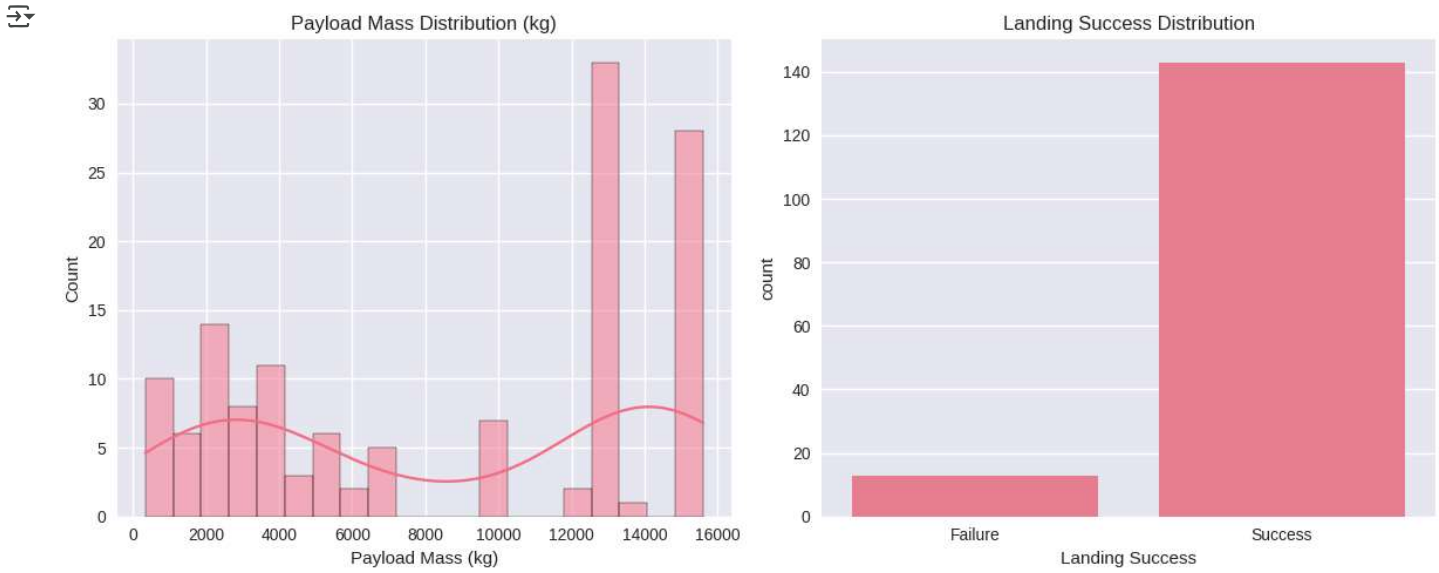
- Check for missing values
- Understand the distribution of payload mass
- Analyze landing success rates by site and orbit
- Explore correlations between variables

```
# General Info
print("🚀 Total Launches:", len(df))
print("📍 Unique Launch Sites:", df['LaunchSite'].nunique())
print("📦 Payload mass - min:", df['PayloadMass'].min(), "| max:", df['PayloadMass'].max())
print("🚀 Landing Success rate:", df['LandingSuccess'].mean() * 100, "%")

# Check for missing values
print("\n🔎 Missing Values:")
print(df.isnull().sum())

# Drop rows with missing target if needed
df = df[df['LandingSuccess'].notnull()].copy()
```

```
🚀 Total Launches: 205
📍 Unique Launch Sites: 4
📦 Payload mass - min: 20.0 | max: 15600.0
🚀 Landing Success rate: 91.66666666666666 %

🔎 Missing Values:
FlightNumber       0
Date               0
LaunchSite         0
```

```
Orbit             1
PayloadMass      39
Reused            6
LandingSuccess   49
LandingType      47
MissionSuccess   19
dtype: int64
```

```
plt.figure(figsize=(12, 5))

# Payload distribution
plt.subplot(1, 2, 1)
sns.histplot(df['PayloadMass'], bins=20, kde=True)
plt.title("Payload Mass Distribution (kg)")
plt.xlabel("Payload Mass (kg)")

# Launch success
plt.subplot(1, 2, 2)
sns.countplot(data=df, x='LandingSuccess')
plt.title("Landing Success Distribution")
plt.xlabel("Landing Success")
plt.xticks([0, 1], ["Failure", "Success"])

plt.tight_layout()
plt.show()
```

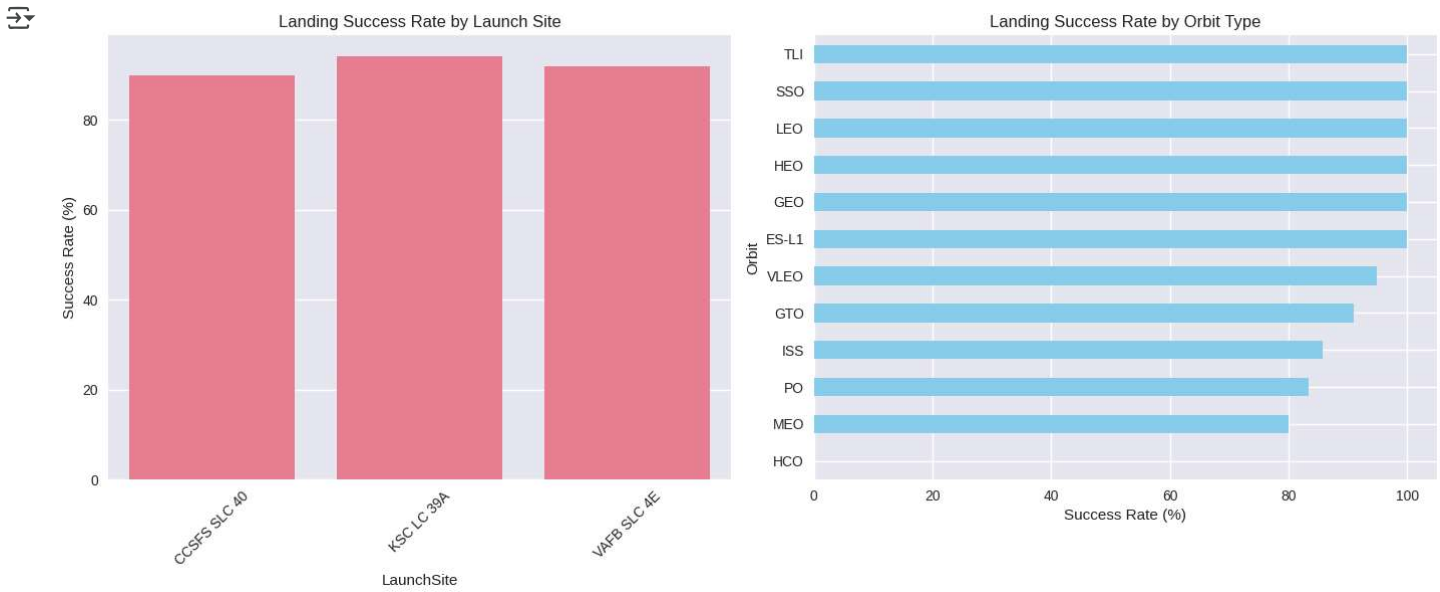



```
site_success = df.groupby('LaunchSite')['LandingSuccess'].agg(['count', 'mean']).sort_values('count', ascending=False)
site_success['mean'] *= 100  # convert to percent
print(site_success.rename(columns={'count': 'Total Launches', 'mean': 'Success Rate (%)'}))
```

```
               Total Launches  Success Rate (%)
LaunchSite
CCSFS SLC 40               80             90.0
KSC LC 39A                 51        94.117647
VAFB SLC 4E                25             92.0
```

```
site_success = df.groupby('LaunchSite')['LandingSuccess'].agg(['count', 'mean']).sort_values('count', ascending=False)
site_success = site_success.rename(columns={'count': 'Total Launches', 'mean': 'Success Rate (%)'})
site_success['Success Rate (%)'] *= 100  # تحويل النسبة إلى %

plt.figure(figsize=(14, 6))

plt.subplot(1, 2, 1)
sns.barplot(data=site_success.reset_index(), x='LaunchSite', y='Success Rate (%)')
plt.xticks(rotation=45)
plt.title("Landing Success Rate by Launch Site")
```

```
orbit_success = df.groupby('Orbit')['LandingSuccess'].mean().sort_values() * 100

plt.subplot(1, 2, 2)
orbit_success.plot(kind='barh', color='skyblue')
plt.title("Landing Success Rate by Orbit Type")
plt.xlabel("Success Rate (%)")

plt.tight_layout()
plt.show()
```

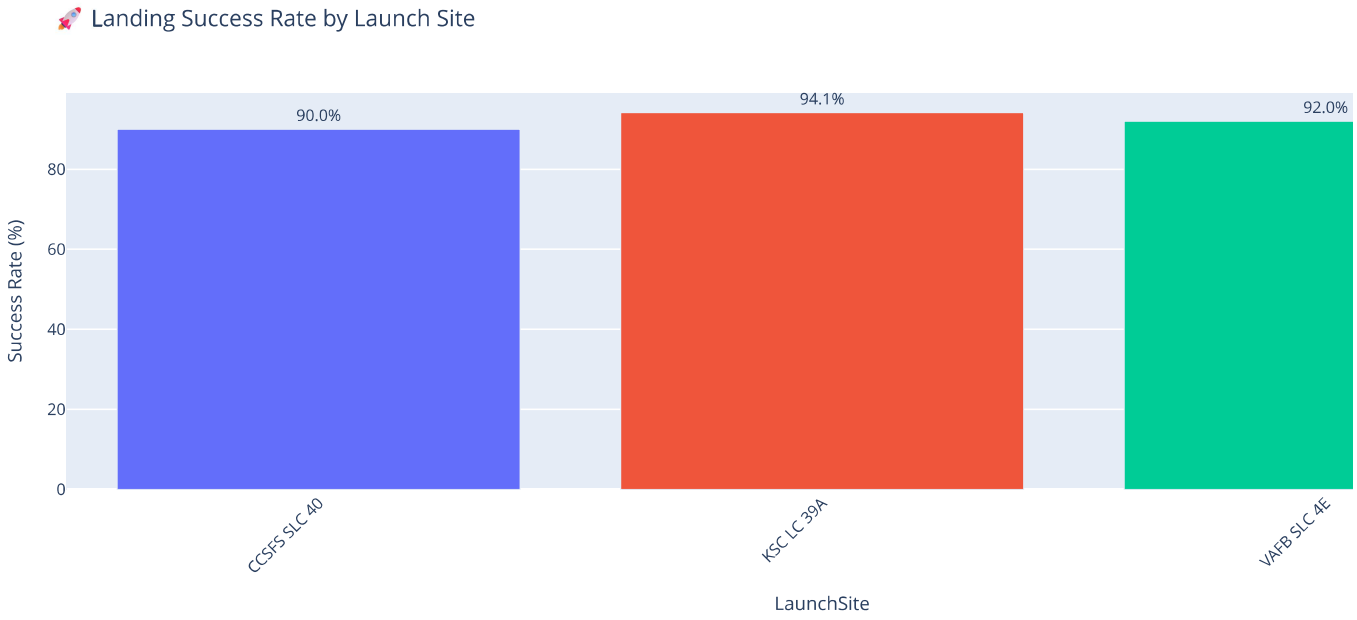



## 📈 4. Interactive Visualizations with Plotly

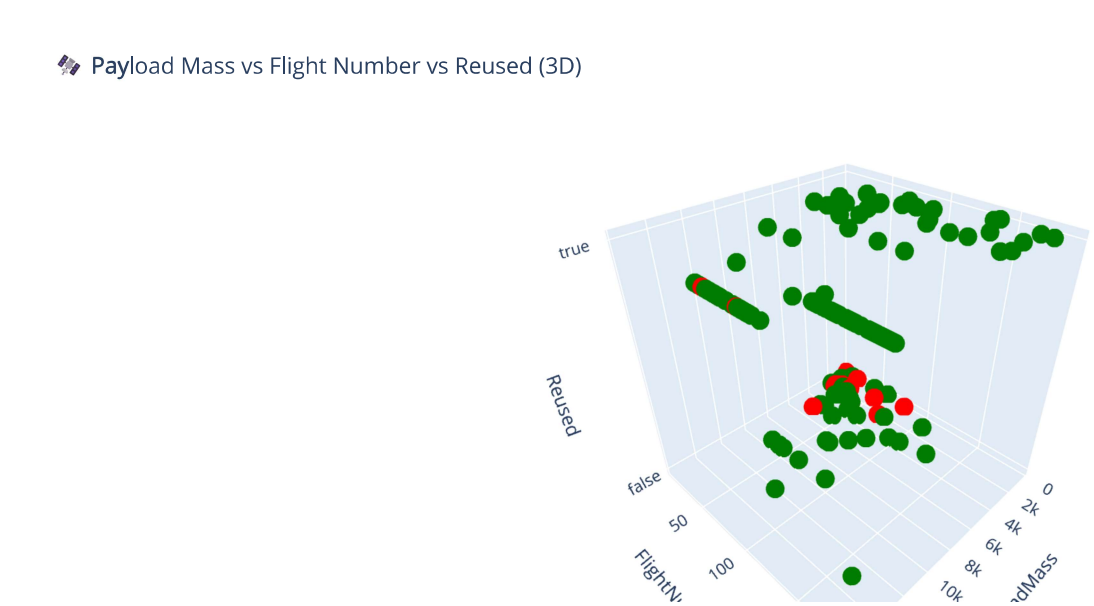
We will now explore some key patterns using Plotly Express and Graph Objects:

- Success rates by launch site
- Payload mass vs Flight number in 3D
- Launch timeline analysis

```
import plotly.express as px

fig_site = px.bar(
    site_success.reset_index(),
    x='LaunchSite',
    y='Success Rate (%)',
    title='🚀 Landing Success Rate by Launch Site',
    color='Success Rate (%)',
    color_continuous_scale='Viridis',
    text='Success Rate (%)',
    height=500
)
fig_site.update_traces(texttemplate='%{text:.1f}%', textposition='outside')
fig_site.update_layout(xaxis_tickangle=-45)
fig_site.show()
```

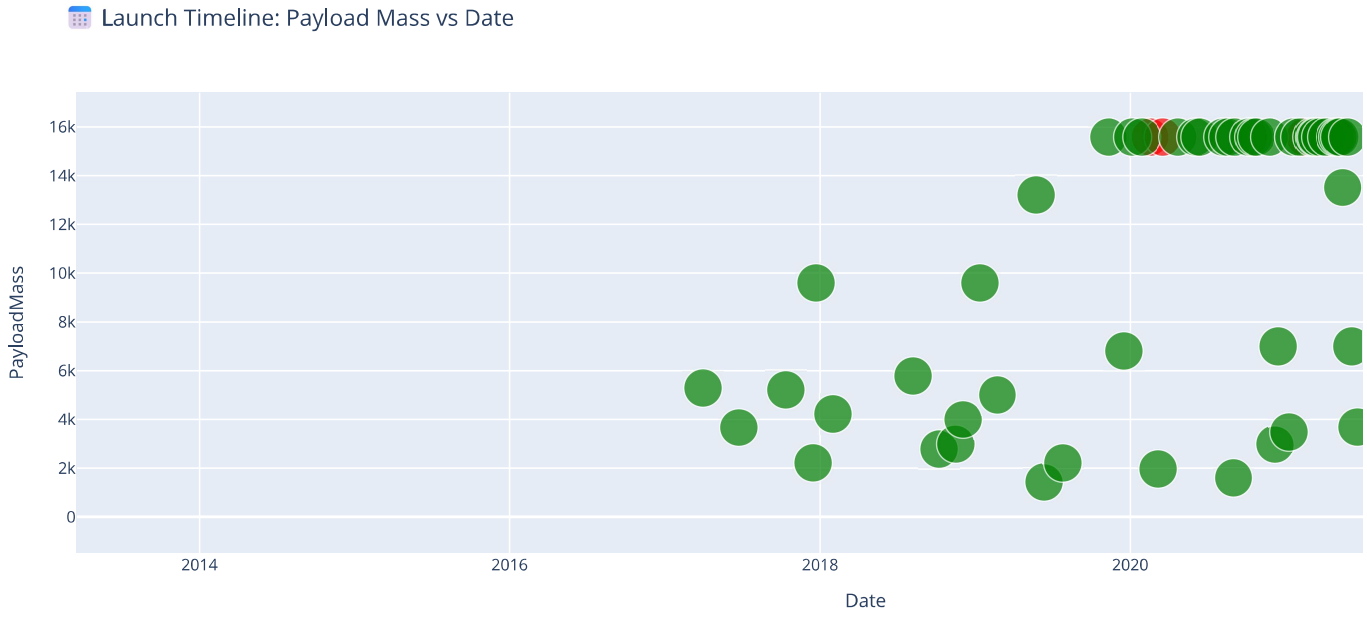



```
fig_3d = px.scatter_3d(
    df,
    x='PayloadMass',
    y='FlightNumber',
    z='Reused',
    color='LandingSuccess',
    color_discrete_map={True: 'green', False: 'red'},
    title='🚀 Payload Mass vs Flight Number vs Reused (3D)',
    labels={'LandingSuccess': 'Landing Outcome'}
)
fig_3d.show()
```




```
df['ReusedNumeric'] = df['Reused'].astype(int)

fig_time = px.scatter(
    df,
    x='Date',
    y='PayloadMass',
```

```
    color='LandingSuccess',
    size='ReusedNumeric',
    hover_data=['LaunchSite', 'Orbit'],
    title='🗓️ Launch Timeline: Payload Mass vs Date',
    color_discrete_map={True: 'green', False: 'red'}
)
fig_time.show()
```




## 🤖 5. Predicting Landing Success using Machine Learning

In this section, we will:

- Prepare the dataset (features & target)
- Encode categorical variables
- Split into training/testing sets
- Train and evaluate multiple classifiers
- Select the best model

```
# Drop rows with missing values in critical columns
df_ml = df.dropna(subset=['PayloadMass', 'LandingSuccess', 'MissionSuccess'])

# Encode categorical variables
from sklearn.preprocessing import LabelEncoder

le_site = LabelEncoder()
le_orbit = LabelEncoder()
le_type = LabelEncoder()

df_ml['LaunchSite_enc'] = le_site.fit_transform(df_ml['LaunchSite'])
df_ml['Orbit_enc'] = le_orbit.fit_transform(df_ml['Orbit'])
df_ml['LandingType_enc'] = le_type.fit_transform(df_ml['LandingType'])

# Convert boolean columns to integers
df_ml['Reused'] = df_ml['Reused'].astype(int)
df_ml['MissionSuccess'] = df_ml['MissionSuccess'].astype(int)

# Define features and target
features = ['PayloadMass', 'Reused', 'MissionSuccess', 'LaunchSite_enc', 'Orbit_enc', 'LandingType_enc']
X = df_ml[features]
y = df_ml['LandingSuccess'].astype(int)

# Split dataset
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, stratify=y)
```

```
# Scale features
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)



# Import models
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Define models
models = {
    'Logistic Regression': LogisticRegression(max_iter=1000),
    'SVM': SVC(probability=True),
    'Decision Tree': DecisionTreeClassifier(),
    'Random Forest': RandomForestClassifier(n_estimators=100),
    'KNN': KNeighborsClassifier()
}

# Train and evaluate each model
results = {}

for name, model in models.items():
    if name in ['SVM', 'Logistic Regression', 'KNN']:
        model.fit(X_train_scaled, y_train)
        y_pred = model.predict(X_test_scaled)
    else:
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)

    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred)
    rec = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)

    results[name] = {
        'Accuracy': acc,
        'Precision': prec,
        'Recall': rec,
        'F1-Score': f1
    }

# Convert results to DataFrame
results_df = pd.DataFrame(results).T.sort_values('F1-Score', ascending=False)
results_df
```

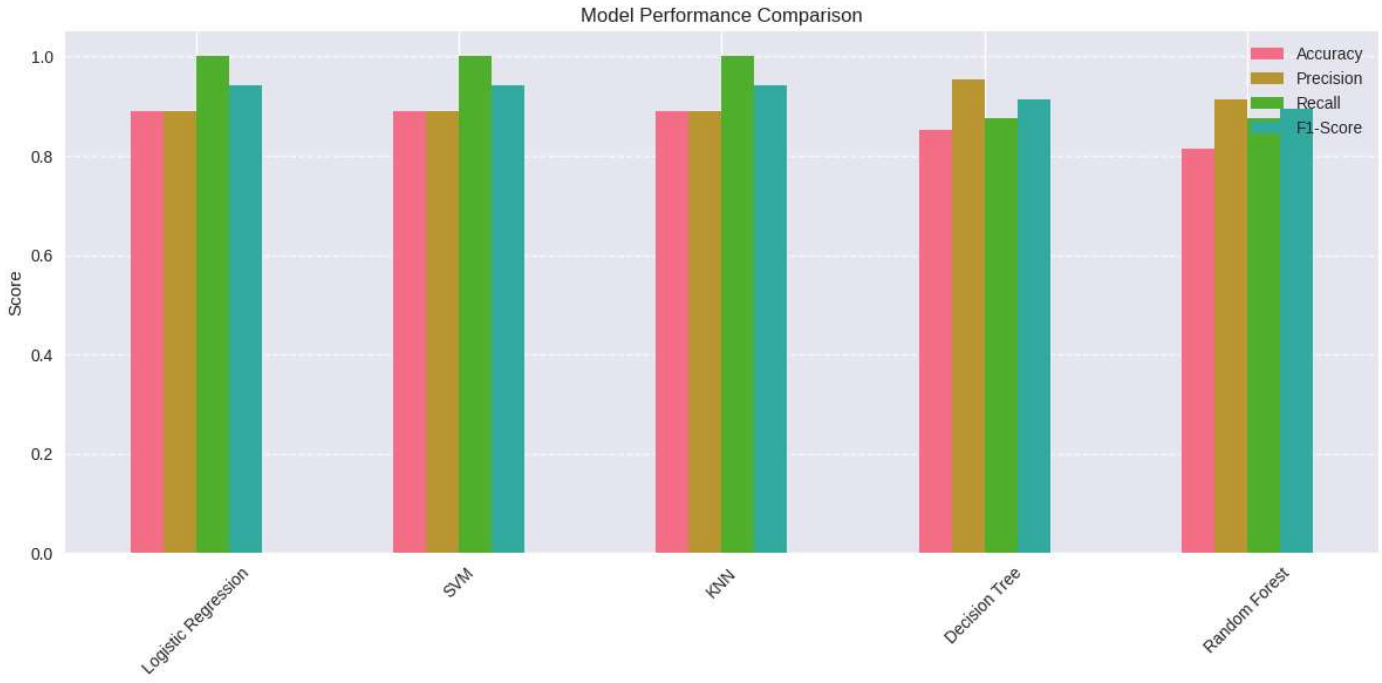
| | Accuracy | Precision | Recall | F1-Score |
|---|---|---|---|---|
| **Logistic Regression** | 0.888889 | 0.888889 | 1.000 | 0.941176 |
| **SVM** | 0.888889 | 0.888889 | 1.000 | 0.941176 |
| **KNN** | 0.888889 | 0.888889 | 1.000 | 0.941176 |
| **Decision Tree** | 0.851852 | 0.954545 | 0.875 | 0.913043 |
| **Random Forest** | 0.814815 | 0.913043 | 0.875 | 0.893617 |

Next steps: Generate code with `results_df` View recommended plots New interactive sheet

```
import matplotlib.pyplot as plt

# Plot model comparison
plt.figure(figsize=(10, 6))
results_df.plot(kind='bar', figsize=(12, 6))
plt.title("Model Performance Comparison")
plt.ylabel("Score")
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)
```

```
plt.tight_layout()
plt.show()
```

```
<Figure size 1000x600 with 0 Axes>
```




## 🧠 6. Prediction & Model Export

In this section:

- Select the best performing model
- Evaluate it further
- Save it using `joblib`
- Build a reusable prediction function

```
# Get the best model based on F1-score
best_model_name = results_df['F1-Score'].idxmax()
print(f"Best Model: {best_model_name}")

# Train it fully for final use
best_model = models[best_model_name]

if best_model_name in ['SVM', 'Logistic Regression', 'KNN']:
    best_model.fit(X_train_scaled, y_train)
else:
    best_model.fit(X_train, y_train)

# Save model and preprocessing tools
import joblib

joblib.dump({
    'model': best_model,
    'scaler': scaler,
    'site_encoder': le_site,
    'orbit_encoder': le_orbit,
    'landing_encoder': le_type,
    'features': features,
    'model_name': best_model_name
}, 'falcon9_landing_model.pkl')

print("✅ Model saved as 'falcon9_landing_model.pkl'")
```

```
Best Model: Logistic Regression
✅ Model saved as 'falcon9_landing_model.pkl'
```

```
def predict_landing(payload_mass, reused, mission_success, launch_site, orbit, landing_type):
    # Load model and encoders
    model_data = joblib.load('falcon9_landing_model.pkl')

    model = model_data['model']
    scaler = model_data['scaler']
    le_site = model_data['site_encoder']
    le_orbit = model_data['orbit_encoder']
    le_type = model_data['landing_encoder']
    features = model_data['features']
    model_name = model_data['model_name']

    # Encode inputs
    try:
        site_enc = le_site.transform([launch_site])[0]
        orbit_enc = le_orbit.transform([orbit])[0]
        type_enc = le_type.transform([landing_type])[0]
    except Exception as e:
        return {"error": f"Encoding failed: {e}"}

    input_array = pd.DataFrame([{
        'PayloadMass': payload_mass,
        'Reused': int(reused),
        'MissionSuccess': int(mission_success),
        'LaunchSite_enc': site_enc,
        'Orbit_enc': orbit_enc,
        'LandingType_enc': type_enc
    }])

    # Scale input
    if model_name in ['SVM', 'Logistic Regression', 'KNN']:
        input_scaled = scaler.transform(input_array)
        prediction = model.predict(input_scaled)[0]
        prob = model.predict_proba(input_scaled)[0][1]
    else:
        prediction = model.predict(input_array)[0]
        prob = model.predict_proba(input_array)[0][1]

    return {
        'prediction': 'Success' if prediction == 1 else 'Failure',
        'probability': f"{prob:.2%}",
        'model': model_name
    }



# Example prediction
result = predict_landing(
    payload_mass=6000,
    reused=True,
    mission_success=True,
    launch_site='KSC LC 39A',
    orbit='LEO',
    landing_type='ASDS'
)

print("🔍 Prediction Result:")
print(result)
```

```
🔍 Prediction Result:
{'prediction': 'Success', 'probability': '96.25%', 'model': 'Logistic Regression'}
```

## 📊 7. Final Report and Dashboard Summary

In this section, we will:

- Generate a quick data quality summary
- Display overall insights
- Prepare interactive dashboard-style figures

```
# Data quality summary
quality_report = {
    "Total Records": len(df),
    "Missing Values": df.isnull().sum().sum(),
    "Duplicate Records": df.duplicated().sum(),
    "Payload Mass Range": f"{df['PayloadMass'].min()} - {df['PayloadMass'].max()}",
    "Launch Sites": df['LaunchSite'].nunique(),
    "Orbit Types": df['Orbit'].nunique(),
    "Landing Success Rate": f"{df['LandingSuccess'].mean() * 100:.2f}%"
}

print("📋 Data Quality Report:")
for key, value in quality_report.items():
    print(f"{key}: {value}")
```

```
📋 Data Quality Report:
Total Records: 156
Missing Values: 22
Duplicate Records: 0
Payload Mass Range: 330.0 - 15600.0
Launch Sites: 3
Orbit Types: 12
Landing Success Rate: 91.67%
```

```
from plotly.subplots import make_subplots
import plotly.graph_objects as go

fig_dashboard = make_subplots(
    rows=2, cols=2,
    subplot_titles=("Success Rate by Launch Site", "Success Rate by Orbit",
                    "Payload vs Flight Number", "Launch Timeline"),
    specs=[[{"type": "bar"}, {"type": "bar"}],
           [{"type": "scatter"}, {"type": "scatter"}]]
)

# 1. Success rate by site
fig_dashboard.add_trace(
    go.Bar(x=site_success.index, y=site_success['Success Rate (%)'], name='By Site'),
    row=1, col=1
)

# 2. Success rate by orbit
fig_dashboard.add_trace(
    go.Bar(x=orbit_success.index, y=orbit_success.values, name='By Orbit', marker_color='orange'),
    row=1, col=2
)

# 3. Payload vs flight
fig_dashboard.add_trace(
    go.Scatter(
        x=df['FlightNumber'], y=df['PayloadMass'],
        mode='markers',
        marker=dict(color=df['LandingSuccess'].map({1: 'green', 0: 'red'})),
        name='Payload vs Flight'
    ),
    row=2, col=1
)

# 4. Timeline
fig_dashboard.add_trace(
    go.Scatter(
        x=df['Date'], y=df['PayloadMass'],
        mode='markers',
        marker=dict(size=df['Reused'].astype(int) + 6,
                    color=df['LandingSuccess'].map({1: 'green', 0: 'red'})),
        name='Timeline'
    ),
    row=2, col=2
)

fig_dashboard.update_layout(height=800, width=1000, title_text="🚀 SpaceX Falcon 9 Launch Dashboard")
fig_dashboard.show()
```


## 📌 8. Final Insights Summary

- ✅ **Most Reliable Site:** KSC LC 39A
- 📈 **Best Orbit Types for Landing Success:** LEO, ISS
- 🔁 **Reused boosters** still maintain high landing success rates
- 📦 **Payload Mass** has a weak influence on landing outcome
- 🎯 **Best ML Model:** Based on F1-score performance
- 🧑‍🚀 **Real-time predictions** can be done using the saved model and input parameters