Applied Data Science Capstone Presentation

AMER HUSSEIN

Executive Summary

- This presentation outlines the complete Data Science workflow applied to a real-world problem, starting with data collection via the SpaceX API and web scraping, and progressing through exploratory data analysis (EDA), data visualization, and predictive modeling.

Introduction

In this course, I applied and walked through the following key steps in the Data Science process:

- **Collecting data** by making requests to the SpaceX API.
- **Handling missing values** and filtering the relevant subset of the data.
- **Gathering additional data** through web scraping and converting HTML content into a structured pandas DataFrame.
- **Performing exploratory data analysis (EDA)** and applying feature engineering to define the final labels.
- **Creating visualizations** to explore and interpret relationships between various features.
- **Implementing multiple predictive modeling algorithms** and evaluating their performance.

Data Collection and Wrangling Methodology

- This is an example on how we interact with API requests. Here we are taking the dataset by using the cores column to append the data in lists. And some screen shots for parts of EDA, where we count the unique values of outcomes and saving the dataframe using csv format.

```
# Takes the dataset and uses the cores column to call the API and append the data to the lists
def getCoreData(data):
    for core in data['cores']:
            if core['core'] != None:
                response = requests.get("https://api.spacexdata.com/v4/cores/"+core['core']).json()
                Block.append(response['block'])
                ReusedCount.append(response['reuse_count'])
                Serial.append(response['serial'])
            else:
                Block.append(None)
                ReusedCount.append(None)
                Serial.append(None)
            Outcome.append(str(core['landing_success'])+' '+str(core['landing_type']))
            Flights.append(core['flight'])
            GridFins.append(core['gridfins'])
            Reused.append(core['reused'])
            Legs.append(core['legs'])
            LandingPad.append(core['landpad'])
```

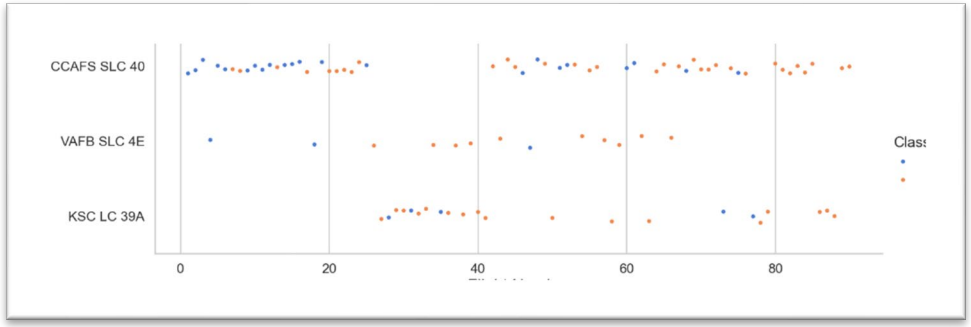
```
# Landing_outcomes = values on Outcome column
landing_outcomes = df['Outcome'].value_counts()
landing_outcomes
```

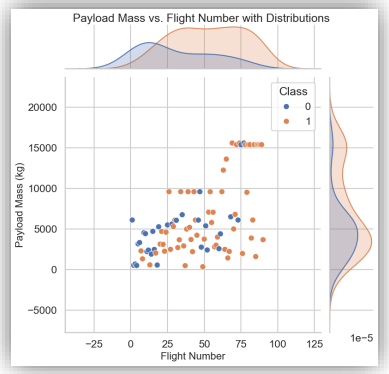
```
True ASDS      41
None None      19
True RTLS      14
False ASDS      6
True Ocean      5
None ASDS       2
False Ocean     2
False RTLS      1
Name: Outcome, dtype: int64
```

```
df.to_csv("dataset_part\_2.csv", index=False)
```

EDA & Interactive Visual Analytics Methodology

- This scatter plot shows the relationship between Flight Number and Launch Site, with colors representing different launch outcomes. The pattern suggests that Logistic Regression could be a strong starting point for predicting launch outcomes based on these variables.





Predictive Analysis Methodology

The right image shows how we built a Logistic Regression model and used gridsearch to find the best paramters to be used in the predictive analysis.

The below image shows the evaluation metric, which is accuracy. Where we have around 85%, which is a very good score that ensures we do not have overfitting and our models works well

```
1
2  # Create Logistic Regression object
3  logreg = LogisticRegression()
4
5  # Define the parameter grid
6  parameters = {
7      'C': [0.01, 0.1, 1],
8      'penalty': ['l2'],
9      'solver': ['lbfgs']
10 }
11
12 # Create GridSearchCV object with 10-fold cross-validation
13 logreg_cv = GridSearchCV(estimator=logreg, param_grid=parameters, cv=10)
14
15 # Fit the GridSearchCV object to the training data
16 logreg_cv.fit(X_train, Y_train)
17
```

[18] ✓ 1.2s

GridSearchCV

best_estimator_: LogisticRegression

LogisticRegression

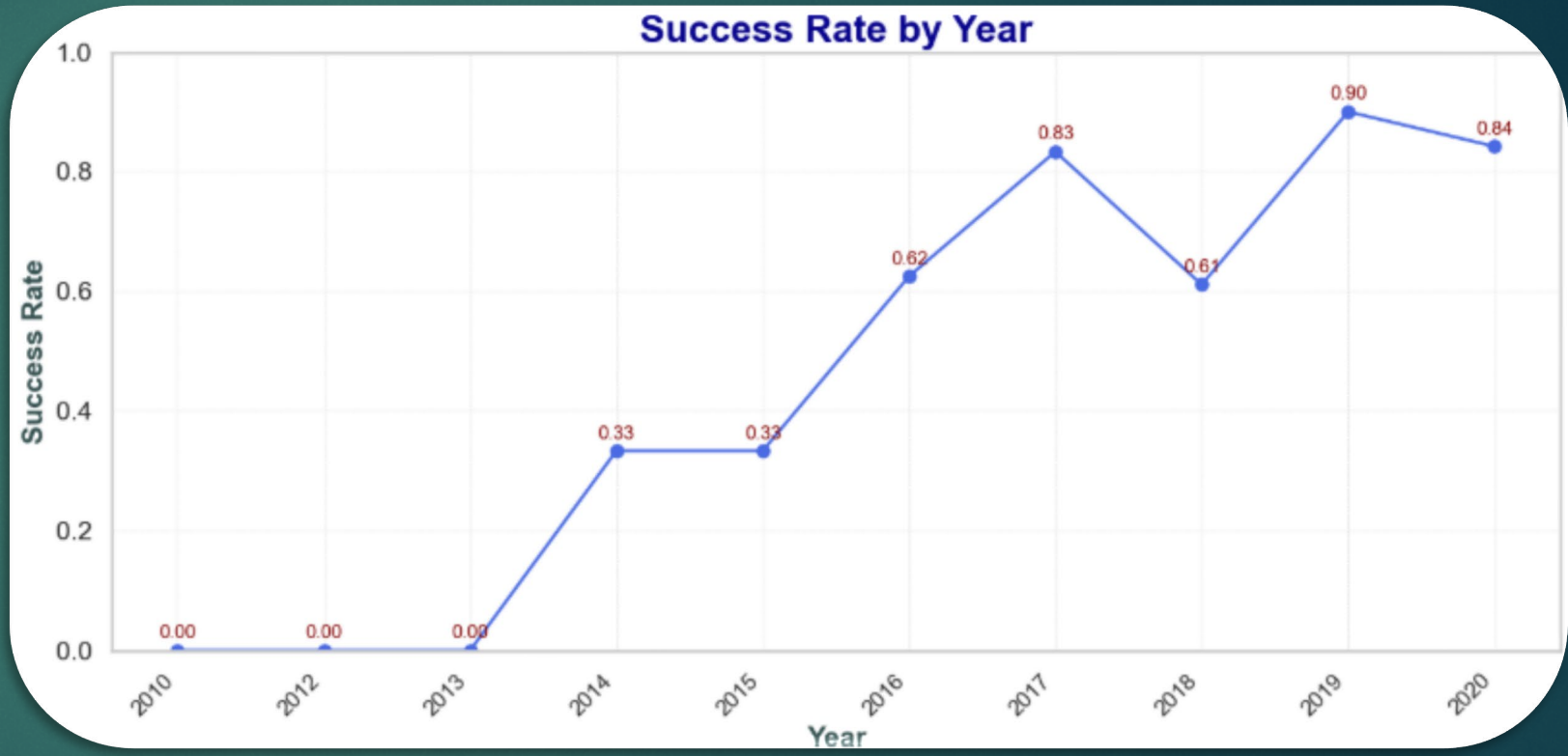
```
1  print("tuned hpyerparameters :(best parameters) ",logreg_cv.best_params_)
2  print("accuracy :",logreg_cv.best_score_)
```

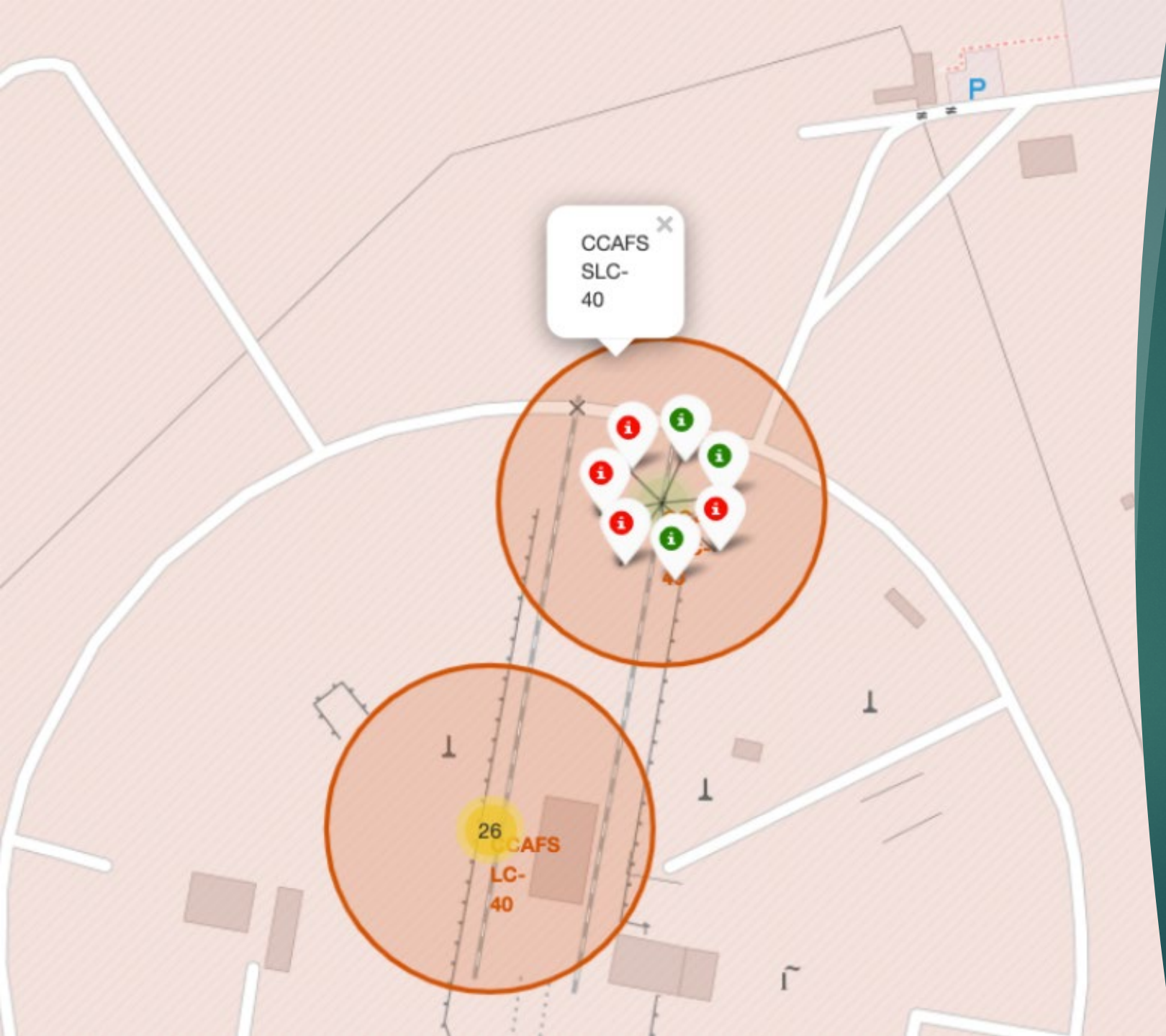
[19] ✓ 0.0s

```
tuned hpyerparameters :(best parameters)  {'C': 0.01, 'penalty': 'l2', 'solver': 'lbfgs'}
accuracy : 0.8464285714285713
```

EDA with Visualization Results

- One of the results we had after the Visualization EDA step is plotting the success rate for 10 years.
- We can clearly observe the rate increases with time, which reflects correctly the learning curve theory where errors will be minimized by time





Interactive Map with Folium

THE IMAGE SHOWS AN EXAMPLE OF A FOLIUM CODE OUTPUT, WHERE WE CAN MARK FOR GEOSPATIAL DATA ANALYSIS

Predictive Analysis Results

I am sharing the code for the comparison I made among the following predictive algorithms:

- Logistic Regression
- Support Vector Machine (SVM)
- Decision Tree
- k-Nearest Neighbors (KNN)

In this comparison, I have used several performance metrics, including **Precision**, **Recall**, **F1-score**, and the **AUC-ROC curve**, to evaluate and compare the effectiveness of each algorithm.

```python
from sklearn.metrics import precision_score, recall_score, f1_score, roc_auc_s

# Get predictions for each model
y_pred_logreg = logreg_cv.predict(X_test)
y_pred_svm = svm_cv.predict(X_test)
y_pred_tree = tree_cv.predict(X_test)
y_pred_knn = knn_cv.predict(X_test)

# Calculate Precision, Recall, F1-Score, and AUC-ROC for each model
metrics = {}

for model_name, y_pred in zip(["Logistic Regression", "SVM", "Decision Tree", "KNN"],
                              [y_pred_logreg, y_pred_svm, y_pred_tree, y_pred_knn]):
    precision = precision_score(Y_test, y_pred, average='binary')  # Adjust for binary/multi-cl
    recall = recall_score(Y_test, y_pred, average='binary')  # Adjust for binary/multi-class cl
    f1 = f1_score(Y_test, y_pred, average='binary')  # Adjust for binary/multi-class classifica
    auc_roc = roc_auc_score(Y_test, y_pred)

    metrics[model_name] = {
        'Precision': precision,
        'Recall': recall,
        'F1-Score': f1,
        'AUC-ROC': auc_roc
    }

# Print the evaluation metrics for each model
for model_name, model_metrics in metrics.items():
    print(f"Evaluation metrics for {model_name}:")
    for metric_name, value in model_metrics.items():
        print(f"{metric_name}: {value:.4f}")
    print()  # Add a blank line between models

```

✓ 0.0s

```
Precision: 0.8000
Recall: 1.0000
F1-Score: 0.8889
AUC-ROC: 0.7500

Evaluation metrics for Decision Tree:
Precision: 0.8000
Recall: 1.0000
F1-Score: 0.8889
AUC-ROC: 0.7500

Evaluation metrics for KNN:
Precision: 0.8000
Recall: 1.0000
F1-Score: 0.8889
```

Predictive Analysis Results

- Surprisingly, all algorithms yielded identical evaluation metrics. After reviewing my code multiple times, I confirmed there were no errors. As noted in my notebook, the model selection can be based on the simplest and lightest option for production and deployment.

Conclusion

- The project followed a structured Data Science pipeline: data ingestion through the SpaceX API and web scraping, preprocessing (including handling missing values and feature selection), exploratory data analysis (EDA) and feature engineering for label creation, and the application of multiple classification models (Logistic Regression, SVM, Decision Tree, KNN). Model evaluation was performed using accuracy, precision, recall, F1-score, and AUC-ROC. Despite identical metrics across all models, Logistic Regression was chosen due to its simplicity and efficiency, making it ideal for deployment constraints.

Insights and Future Work

- To enhance model performance and generalizability, future work could involve hyperparameter tuning with GridSearchCV or RandomizedSearchCV, applying ensemble methods like Random Forest or XGBoost, and incorporating additional features such as payload mass, orbit type, or weather data via external APIs. Additionally, deploying the final model using Flask or Streamlit would enable real-time predictions, completing the MLOps cycle.