

Final Project

<https://www.kaggle.com/datasets/thirumani/shark-tank-us-dataset>

Overview: This dataset contains 53 columns of information about 1,460+ Shark Tank pitches from season 1 to season 17. Shark Tank is a popular TV show where aspiring entrepreneurs pitch their business ideas to a panel of wealthy investors. It contains data such as information about the pitchers, Sharks present, the company pitching, and the investment deal. It is useful to see why a startup receives or fails to secure an investment from the Sharks.

Categorical attributes:

- Season Number
- Startup Name
- Episode Number
- Pitch Number
- Season Start
- Season End
- Original Air Date
- Industry
- Business Description
- Company Website
- Pitchers Gender
- Pitchers City
- Pitchers State
- Pitchers Average Age
- Entrepreneur Names
- Multiple Entrepreneurs
- Got Deal
- Royalty Deal
- Deal has conditions
- Guest Name
- Barbara Corcoran Present
- Mark Cuban Present
- Lori Greiner Present
- Robert Herjavec Present
- Daymond John Present
- Kevin O Leary Present
- Guest Present

Numerical attributes:

- US Viewership
- Original Ask Amount
- Original Offered Equity
- Valuation Requested
- Total Deal Amount
- Total Deal Equity
- Deal Valuation
- Number of sharks in deal
- Investment Amount Per Shark
- Equity Per Shark
- Advisory Shares Equity
- Loan
- Barbara Corcoran Investment Amount
- Barbara Corcoran Investment Equity
- Mark Cuban Investment Amount
- Mark Cuban Investment Equity
- Lori Greiner Investment Amount
- Lori Greiner Investment Equity
- Robert Herjavec Investment Amount
- Robert Herjavec Investment Equity
- Daymond John Investment Amount
- Daymond John Investment Equity
- Kevin O Leary Investment Amount
- Kevin O Leary Investment Equity
- Guest Investment Amount
- Guest Investment Equity
- Season Start
- Season End

Use Case: This dataset poses a classification problem of predicting whether a startup receives an investment based on pitch and founder attributes.

Requirement 1

I used `.info()` to get a concise summary of my data, including each column's count and data type:

Data columns (total 53 columns):			
#	Column	Non-Null Count	Dtype
0	Season Number	1465 non-null	int64
1	Startup Name	1465 non-null	object
2	Episode Number	1465 non-null	int64
3	Pitch Number	1465 non-null	int64
4	Season Start	1465 non-null	object
5	Season End	1441 non-null	object
6	Original Air Date	1465 non-null	object
7	Industry	1465 non-null	object
8	Business Description	1465 non-null	object
9	Company Website	941 non-null	object
10	Pitchers Gender	1458 non-null	object
11	Pitchers Average Age	529 non-null	object
12	Pitchers City	809 non-null	object
13	Pitchers State	936 non-null	object
14	Entrepreneur Names	970 non-null	object
15	Multiple Entrepreneurs	1038 non-null	float64
16	US Viewership	1465 non-null	float64
17	Original Ask Amount	1465 non-null	int64
18	Original Offered Equity	1465 non-null	float64
19	Valuation Requested	1465 non-null	int64
20	Got Deal	1465 non-null	int64
21	Total Deal Amount	900 non-null	float64
22	Total Deal Equity	900 non-null	float64
23	Deal Valuation	900 non-null	float64
24	Number of Sharks in Deal	900 non-null	float64
25	Investment Amount Per Shark	900 non-null	float64
26	Equity Per Shark	900 non-null	float64
27	Royalty Deal	89 non-null	object
28	Advisory Shares Equity	4 non-null	float64
29	Loan	59 non-null	float64
30	Deal Has Conditions	3 non-null	object
31	Barbara Corcoran Investment Amount	144 non-null	float64
32	Barbara Corcoran Investment Equity	144 non-null	float64
33	Mark Cuban Investment Amount	263 non-null	float64
34	Mark Cuban Investment Equity	263 non-null	float64
35	Lori Greiner Investment Amount	239 non-null	float64
36	Lori Greiner Investment Equity	239 non-null	float64
37	Robert Herjavec Investment Amount	136 non-null	float64
38	Robert Herjavec Investment Equity	136 non-null	float64
39	Daymond John Investment Amount	125 non-null	float64
40	Daymond John Investment Equity	125 non-null	float64
41	Kevin O Leary Investment Amount	135 non-null	float64
42	Kevin O Leary Investment Equity	135 non-null	float64
43	Guest Investment Amount	144 non-null	float64
44	Guest Investment Equity	144 non-null	float64
45	Guest Name	144 non-null	object
46	Barbara Corcoran Present	994 non-null	float64
47	Mark Cuban Present	1064 non-null	float64
48	Lori Greiner Present	1092 non-null	float64
49	Robert Herjavec Present	980 non-null	float64
50	Daymond John Present	990 non-null	float64
51	Kevin O Leary Present	1089 non-null	float64
52	Guest Present	127 non-null	float64
dtypes: float64(32), int64(6), object(15)			
memory usage: 606.7+ KB			

Next, I printed the value counts of the “Industry” column and received the following results:

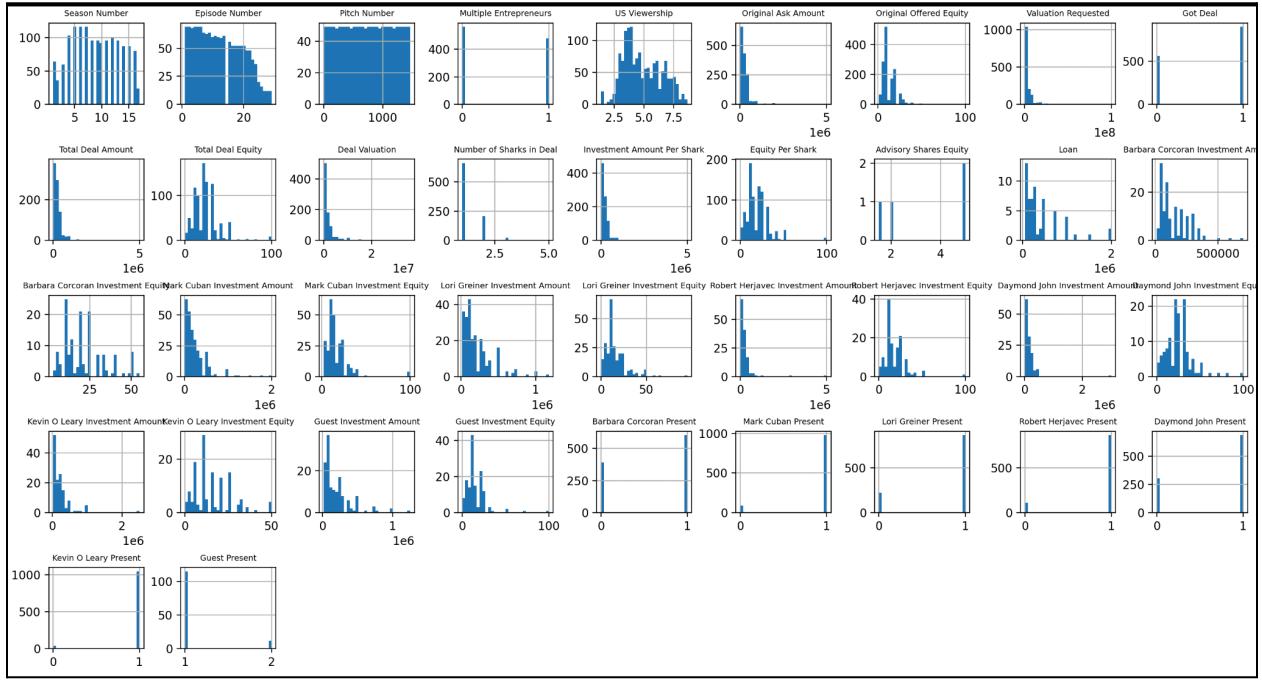
Industry	
Food and Beverage	318
Lifestyle/Home	258
Fashion/Beauty	238
Fitness/Sports/Outdoors	150
Children/Education	134
Health/Wellness	70
Technology/Software	69
Pet Products	63
Business Services	42
Media/Entertainment	28
Uncertain/Other	23
Electronics	18
Automotive	17
Green/CleanTech	14
Liquor/Alcohol	12
Travel	11
Name: count, dtype: int64	

I then used describe() to get statistics on the dataset's tendencies and distribution:

	Season Number	Episode Number	Pitch Number	Multiple Entrepreneurs	...	Robert Herjavec Present	Daymond John Present	Kevin O Leary Present	Guest Present
count	1465.000000	1465.000000	1465.000000	1038.000000	...	980.000000	990.000000	1089.000000	127.000000
mean	8.930375	12.203413	733.000000	0.459538	...	0.885714	0.692929	0.963269	1.094488
std	4.351598	7.374514	423.053385	0.498600	...	0.318320	0.461512	0.188187	0.293665
min	1.000000	1.000000	1.000000	0.000000	...	0.000000	0.000000	0.000000	1.000000
25%	5.000000	6.000000	367.000000	0.000000	...	1.000000	0.000000	1.000000	1.000000
50%	9.000000	12.000000	733.000000	0.000000	...	1.000000	1.000000	1.000000	1.000000
75%	13.000000	18.000000	1099.000000	1.000000	...	1.000000	1.000000	1.000000	1.000000
max	17.000000	29.000000	1465.000000	1.000000	...	1.000000	1.000000	1.000000	2.000000

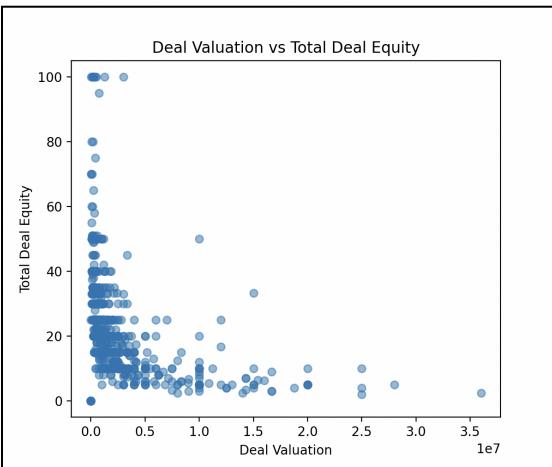
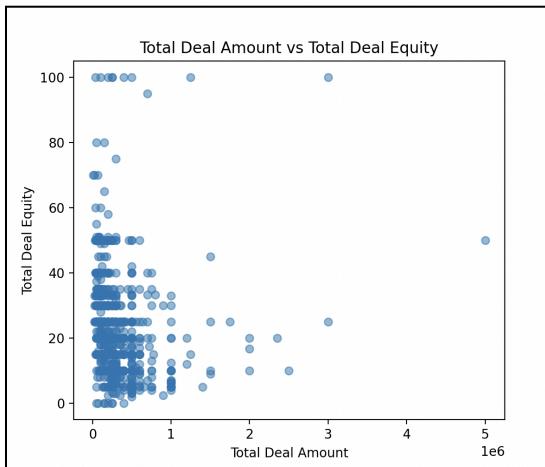
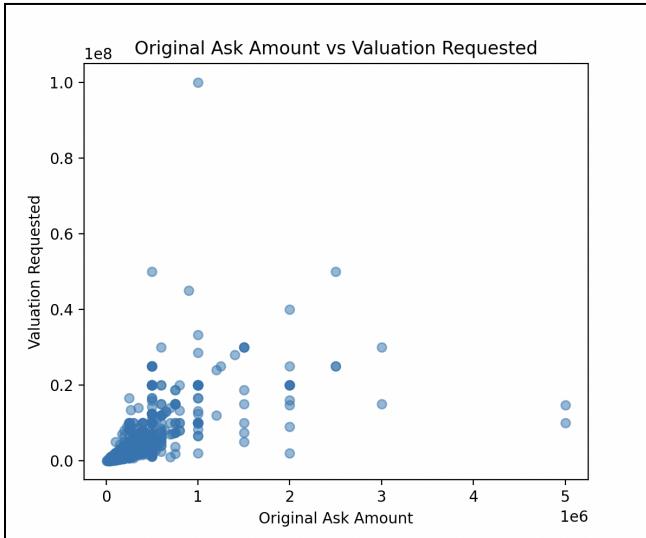
[8 rows x 38 columns]

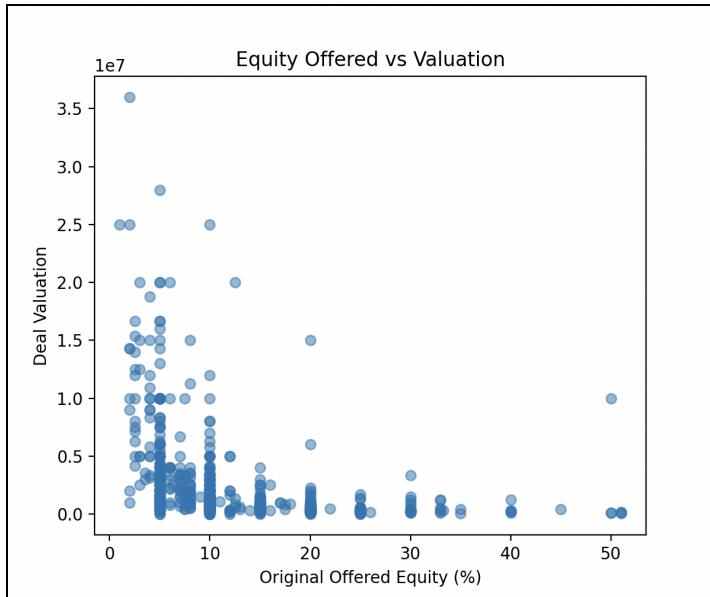
Using the hist() and plt() functions, I visualized all of my data. However, with the number of numerical columns (38), I had to restructure my plt() function for a tighter layout and smaller title sizes.



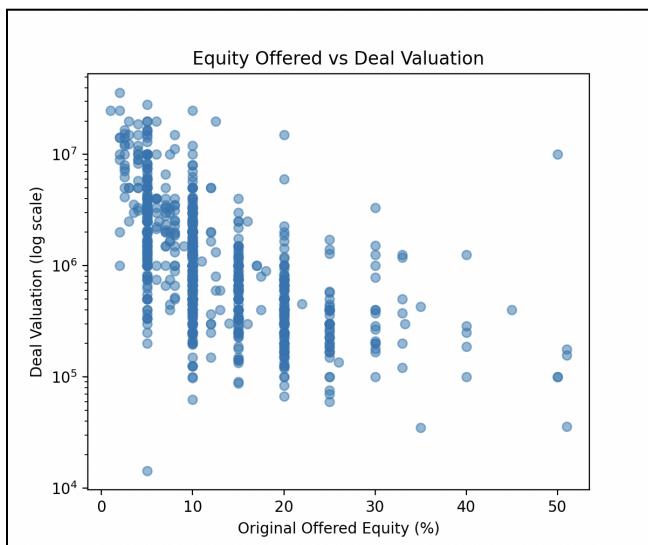
Requirement 2

I generated 5 scatterplots to visualize numeric relationships within my dataset:





The previous result looked off due to skew and outliers, so I used a log scale to show off the relationship better:



Requirement 3

I used a simple corr() matrix to see linear relationships between my numeric data and received this output:

memory usage: 606.7+ KB	
Got Deal	1.000000
Season Number	0.170425
Pitch Number	0.169561
Lori Greiner Present	0.140779
Multiple Entrepreneurs	0.129470
Mark Cuban Present	0.080436
Guest Present	0.050320
Barbara Corcoran Present	-0.000269
Episode Number	-0.007052
Kevin O Leary Present	-0.023647
Valuation Requested	-0.036221
Robert Herjavec Present	-0.042456
Daymond John Present	-0.052090
Original Ask Amount	-0.081039
Original Offered Equity	-0.111943
US Viewership	-0.125000
Total Deal Amount	NaN
Total Deal Equity	NaN
Deal Valuation	NaN
Number of Sharks in Deal	NaN
Investment Amount Per Shark	NaN
Equity Per Shark	NaN
Advisory Shares Equity	NaN
Loan	NaN
Barbara Corcoran Investment Amount	NaN
Barbara Corcoran Investment Equity	NaN
Mark Cuban Investment Amount	NaN
Mark Cuban Investment Equity	NaN
Lori Greiner Investment Amount	NaN
Lori Greiner Investment Equity	NaN
Robert Herjavec Investment Amount	NaN
Robert Herjavec Investment Equity	NaN
Daymond John Investment Amount	NaN
Daymond John Investment Equity	NaN
Kevin O Leary Investment Amount	NaN
Kevin O Leary Investment Equity	NaN
Guest Investment Amount	NaN
Guest Investment Equity	NaN
Name: Got Deal, dtype: float64	

Of the different columns, there doesn't seem to be a strong relationship between any of the values. I believe modifications to the data are needed.

Note: I want to fix the NaN values (which I believe come from missing values) and plan to clean my data in the next step.

Requirement 4

Since I am evaluating what leads to the investment deal, I am eliminating the post-deal values, as well as categories with a majority of null values or that were irrelevant to the deal:

- Total Deal Amount
- Total Deal Equity
- Deal Valuation
- Number of sharks in deal
- Investment Amount Per Shark
- Equity Per Shark
- Advisory Shares Equity
- Loan
- Barbara Corcoran Investment Amount
- Barbara Corcoran Investment Equity
- Mark Cuban Investment Amount
- Mark Cuban Investment Equity
- Lori Greiner Investment Amount
- Lori Greiner Investment Equity
- Robert Herjavec Investment Amount
- Robert Herjavec Investment Equity
- Daymond John Investment Amount
- Daymond John Investment Equity
- Kevin O Leary Investment Amount
- Kevin O Leary Investment Equity
- Guest Investment Amount
- Guest Investment Equity
- Guest Name
- Guest Present

```
#Requirement 4
leakage_columns = [
    "Total Deal Amount",
    "Total Deal Equity",
    "Deal Valuation",
    "Number of sharks in deal",
    "Investment Amount Per Shark",
    "Equity Per Shark",
    "Advisory Shares Equity",
    "Loan",
    "Barbara Corcoran Investment Amount",
    "Barbara Corcoran Investment Equity",
    "Mark Cuban Investment Amount",
    "Mark Cuban Investment Equity",
    "Lori Greiner Investment Amount",
    "Lori Greiner Investment Equity",
    "Robert Herjavec Investment Amount",
    "Robert Herjavec Investment Equity",
    "Daymond John Investment Amount",
    "Daymond John Investment Equity",
    "Kevin O Leary Investment Amount",
    "Kevin O Leary Investment Equity",
    "Guest Investment Amount",
    "Guest Investment Equity"
]

shark_prep = shark.drop(columns=leakage_columns, errors="ignore")

target = shark["Got Deal"]
```

Next, I used a pipeline for my categorical and numerical columns, where numerical values were imputed using the median and categorical values were imputed using the most frequent values and OneHotEncoder. Finally, I used a ColumnTransformer to apply these preoccesing steps to my columns.

```
categorical_features = [
    "Industry",
    "Pitchers Gender",
    "Pitchers State",
    "Multiple Entrepreneurs",
    "Barbara Corcoran Present",
    "Mark Cuban Present",
    "Lori Greiner Present",
    "Robert Herjavec Present",
    "Daymond John Present",
    "Kevin O Leary Present",
    "Guest Present"
]

numerical_features = [
    "Season Number",
    "Episode Number",
    "Pitch Number",
    "US Viewership",
    "Original Ask Amount",
    "Original Offered Equity",
    "Valuation Requested"
]

num_pipeline = Pipeline([
    ("imputer", SimpleImputer(strategy="median")),
    ("scaler", StandardScaler())
])

cat_pipeline = Pipeline([
    ("imputer", SimpleImputer(strategy="most_frequent")),
    ("encoder", OneHotEncoder(handle_unknown="ignore"))
])

preprocessor = ColumnTransformer([
    ("num", num_pipeline, numerical_features),
    ("cat", cat_pipeline, categorical_features)
])

shark_prepared = preprocessor.fit_transform(shark_prep)
```

Requirement 5

Because my dataset is a classification problem, I'm choosing to use Logistic Regression (as it has a simple, interpretable baseline) and Random Forest Classifier (non-linear, has a higher capacity model). After importing the scikit models and preprocessing my pipelines, I performed cross-validation:

```
#Requirement 5
log_reg_pipeline = Pipeline([
    ("preprocess", preprocess),
    ("model", LogisticRegression(max_iter=1000))
])
    (variable) preprocess: ColumnTransformer
rf_pipeline = Pipeline([
    ("preprocess", preprocess),
    ("model", RandomForestClassifier(
        n_estimators=100,
        random_state=42
    ))
])
log_reg_scores = cross_val_score(
    log_reg_pipeline,
    shark_prep,
    target,
    cv=5,
    scoring="accuracy"
)

rf_scores = cross_val_score(
    rf_pipeline,
    shark_prep,
    target,
    cv=5,
    scoring="accuracy"
)

print("Logistic Regression CV Accuracy:", log_reg_scores.mean())
print("Random Forest CV Accuracy:", rf_scores.mean())
```

And received these scores:

```
Logistic Regression CV Accuracy: 0.5146757679180888
Random Forest CV Accuracy: 0.3119453924914676
```

I thought these scores were low, so I checked how many of the pitches actually received a deal to see what the minimum accuracy for my model should be:

Got Deal
1 0.614334
0 0.385666

Seeing this, my accuracy is worse than my baseline. Thus, I know something is wrong with my code. I ended up switching my scoring process to F1 to better assess scores across both classes.

```
log_reg_pipeline = Pipeline([
    ("preprocess", preprocessor),
    ("model", LogisticRegression(max_iter=1000, class_weight="balanced"))
])

rf_pipeline = Pipeline([
    ("preprocess", preprocessor),
    ("model", RandomForestClassifier(n_estimators=100, random_state=42))
])

log_reg_f1_scores = cross_val_score(
    log_reg_pipeline, shark_prep, target, cv=5, scoring="f1"
)

rf_f1_scores = cross_val_score(
    rf_pipeline, shark_prep, target, cv=5, scoring="f1"
)

print("LogReg (balanced) CV F1:", log_reg_f1_scores.mean())
print("RandomForest CV F1:", rf_f1_scores.mean())
```

However, my accuracy was still low:

```
LogReg (balanced) CV F1: 0.43621127402214055
RandomForest CV F1: 0.29952460362315303
```

After some research, I am content with these scores as they are a more realistic result of the model trying to catch both deals and no-deals rather than just deals. With fair evaluation, LogisticRegression performs better than RandomForest.

Requirement 6

In this part, I am running a GridSearchCV to find the best hyperparameters for my model, starting with a broad search to find the best regularization region using two solvers and different scales of regularization:

```
param_grid_1 = {
    "model_solver": ["liblinear", "lbfgs"],
    "model_C": [0.001, 0.01, 0.1, 1, 10, 100],
    "model_penalty": ["l2"]
}

grid_1 = GridSearchCV(
    log_reg_pipeline,
    param_grid_1,
    cv=5,
    scoring="f1",
    n_jobs=-1
)

grid_1.fit(shark_prep, target)
print("Search 1 best params:", grid_1.best_params_)
print("Search 1 best F1:", grid_1.best_score_)
```

My results:

```
Search 1 best params: {'model_C': 0.001, 'model_penalty': 'l2', 'model_solver': 'liblinear'}
Search 1 best F1: 0.45421009479206625
```

Between a small C and L2 penalty, this shows me that the model performs best when the regularization is strong. Also, the liblinear solver is best for smaller datasets (mine is only about 1500 rows).

I then went on to my second search, focusing on the area around my C value:

```
param_grid_2 = {  
    "model__solver": ["liblinear"],  
    "model__penalty": ["l2"],  
    "model__C": [0.0001, 0.0005, 0.001, 0.005, 0.01]  
}  
  
grid_2 = GridSearchCV(  
    log_reg_pipeline,  
    param_grid_2,  
    cv=5,  
    scoring="f1",  
    n_jobs=-1  
)  
  
grid_2.fit(shark_prep, target)  
  
print("Search 2 best params:", grid_2.best_params_)  
print("Search 2 best F1:", grid_2.best_score_)
```

Output:

```
Search 2 best params: {'model__C': 0.0001, 'model__penalty': 'l2', 'model__solver': 'liblinear'}  
Search 2 best F1: 0.4688435642783234
```

My F1 slightly increased, and my best C value went even lower, reaffirming that strong regularization is the best. Upon further research, I also learned that this meant that no single startup feature strongly predicts a deal, but rather it is a holistic interpretation.

For my final search, I centered around my C value:

```
param_grid_3 = {  
    "model__solver": ["liblinear"],  
    "model__penalty": ["l1", "l2"],  
    "model__C": [0.00005, 0.0001, 0.0002]  
}  
  
grid_3 = GridSearchCV(  
    log_reg_pipeline,  
    param_grid_3,  
    cv=5,  
    scoring="f1",  
    n_jobs=-1  
)  
  
grid_3.fit(shark_prep, target)  
  
print("Search 3 best params:", grid_3.best_params_)  
print("Search 3 best F1:", grid_3.best_score_)
```

Output:

```
Search 3 best params: {'model__C': 5e-05, 'model__penalty': 'l2', 'model__solver': 'liblinear'}  
Search 3 best F1: 0.4703618038293499
```

As expected, my F1 score increased while my best C decreased even further. I now understand that overfitting can happen very easily in my model, and that a holistic review is necessary for a good prediction rather than relying on one variable.

Requirement 7

I created a test training set to compare against my previous models and scores:

```
shark_train, shark_test, target_train, target_test = train_test_split(  
    shark_prep,  
    target,  
    test_size=0.2,  
    random_state=42,  
    stratify=target  
)  
  
final_model.fit(shark_train, target_train)  
  
target_pred = final_model.predict(shark_test)  
  
test_f1 = f1_score(target_test, target_pred)  
print("Test F1-score:", test_f1)  
  
print("\nClassification Report:")  
print(classification_report(target_test, target_pred))
```

Output:

```
Test F1-score: 0.6073619631901841  
  
Classification Report:  
             precision      recall   f1-score  support  
          0       0.45      0.58      0.51      113  
          1       0.68      0.55      0.61      180  
  
      accuracy           0.56      293  
  macro avg       0.56      0.57      0.56      293  
weighted avg       0.59      0.56      0.57      293
```

Requirement 8

From my results, I can see my test set is better at predicting deals that happened versus those that didn't. I also received a higher F1 score, exceeding my CV scores and indicating strong generalization. Overall, the results suggest that the model does not overfit and captures meaningful (but limited) predictive signals from pre-pitch features.

My conclusion is that while pre-pitch features such as those in this dataset are key for understanding what goes into a successful investment deal, investor pitches are inherently subjective and require more complexity than what is available in this dataset. Strong regularization and careful evaluation helped ensure realistic performance estimates, but highlights both the potential and the limitations of using structured data to model entrepreneurial decision-making.