First we load the dataset zip file and unzip it.

In [ ]:
```
!unzip /content/shark_tank_data.zip
```

Then we use pandas library to get read the csv file

In [ ]:
```
import pandas as pd

df = pd.read_csv('Shark Tank US dataset.csv')
```

In [ ]:
```
df.head()
```

In [ ]:
```
num_rows = len(df)
print("Number of rows:", num_rows)
```

```
Number of rows: 1365
```

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

Below we can see all the column names of the dataset

```
Index(['Season Number', 'Startup Name', 'Episode Number', 'Pitch Number',
       'Season Start', 'Season End', 'Original Air Date', 'Industry',
       'Business Description', 'Company Website', 'Pitchers Gender',
       'Pitchers Average Age', 'Pitchers City', 'Pitchers State',
       'Entrepreneur Names', 'Multiple Entrepreneurs', 'US Viewership',
       'Original Ask Amount', 'Original Offered Equity', 'Valuation Requested',
       'Got Deal', 'Total Deal Amount', 'Total Deal Equity', 'Deal Valuation',
       'Number of Sharks in Deal', 'Investment Amount Per Shark',
       'Equity Per Shark', 'Royalty Deal', 'Advisory Shares Equity', 'Loan',
       'Deal Has Conditions', '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', 'Barbara Corcoran Present',
       'Mark Cuban Present', 'Lori Greiner Present', 'Robert Herjavec Present',
       'Daymond John Present', 'Kevin O Leary Present', 'Guest Present'],
      dtype='object')
```

# General Information about the Show

Season Number: Identifies the season of Shark Tank; useful for tracking trends across seasons.

Startup Name: Name of the startup; not directly useful for prediction but can help with descriptive analysis.

Episode Number: Episode where the pitch aired; helpful to see if episode positioning affects deals.

Pitch Number: Order of the pitch in the episode; can be useful for determining any patterns related to pitch order.

Season Start / Season End: Start and end dates of the season; useful for understanding time-based trends.

Original Air Date: The date the pitch aired; might affect viewership or investment based on seasonal factors.

# Business and Pitch Information

Industry: The industry of the startup (e.g., tech, food, etc.); a key feature for predicting investment preferences.

Business Description: Description of the business; qualitative data that could be processed using NLP for insights.

Company Website: URL of the startup's website; not directly useful for prediction unless analyzed for metadata.

# Entrepreneur Information

Pitchers Gender: Gender(s) of the entrepreneurs; important for studying gender biases.

Pitchers Average Age: Average age of the pitching team; might correlate with investment likelihood.

Pitchers City: City where the entrepreneurs are based; location-based trends might exist.

Pitchers State: State where the entrepreneurs are based; could help analyze geographic preferences.

Entrepreneur Names: Names of the entrepreneurs; likely not useful for prediction but helpful for identification.

Multiple Entrepreneurs: Indicates whether multiple entrepreneurs are pitching; team dynamics might influence investments.

# Show Metrics

US Viewership: Number of viewers for the episode; higher viewership might lead to higher deal likelihood or value.

# Investment and Financial Details

Original Ask Amount: The amount of money the entrepreneurs initially requested; a critical predictor.

Original Offered Equity: Percentage of equity offered by the entrepreneurs; important for deal valuation.

Valuation Requested: Implied valuation based on the ask and equity offered; key financial metric.

Got Deal: Whether the startup got a deal (yes/no); target variable for prediction.

Total Deal Amount: Amount of money agreed upon in the deal; outcome variable for deal value prediction.

Total Deal Equity: Equity agreed upon in the deal; critical for understanding shark preferences.

Deal Valuation: Valuation of the company based on the final deal; another key outcome metric.

Number of Sharks in Deal: Number of sharks involved; indicates collaboration among investors.

Investment Amount Per Shark: Average amount invested by each shark; useful for analyzing shark-specific behavior.

Equity Per Shark: Average equity granted to each shark; highlights equity preferences.

# Deal-Specific Features

Royalty Deal: Indicates if the deal involved royalties; might affect deal likelihood.

Advisory Shares Equity: If advisory shares were part of the deal; a factor in unique deals.

Loan: Indicates whether the deal involved a loan; useful for identifying alternative deal structures.

Deal Has Conditions: Whether the deal was conditional; might affect the likelihood of deals closing.

# Shark-Specific Investments

Barbara Corcoran Investment Amount: Amount invested by Barbara; indicates her investment patterns.

Barbara Corcoran Investment Equity: Equity Barbara received; shows her equity preferences.

Mark Cuban Investment Amount: Similar metrics for Mark Cuban.

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: Amount invested by guest sharks; tracks guest involvement.

Guest Investment Equity

Guest Name: Name of the guest shark; not directly useful unless analyzing individual guest preferences.

# Shark Presence

Barbara Corcoran Present: Whether Barbara was present in the episode; might affect deal likelihood.

Mark Cuban Present: Same metric for Mark Cuban.

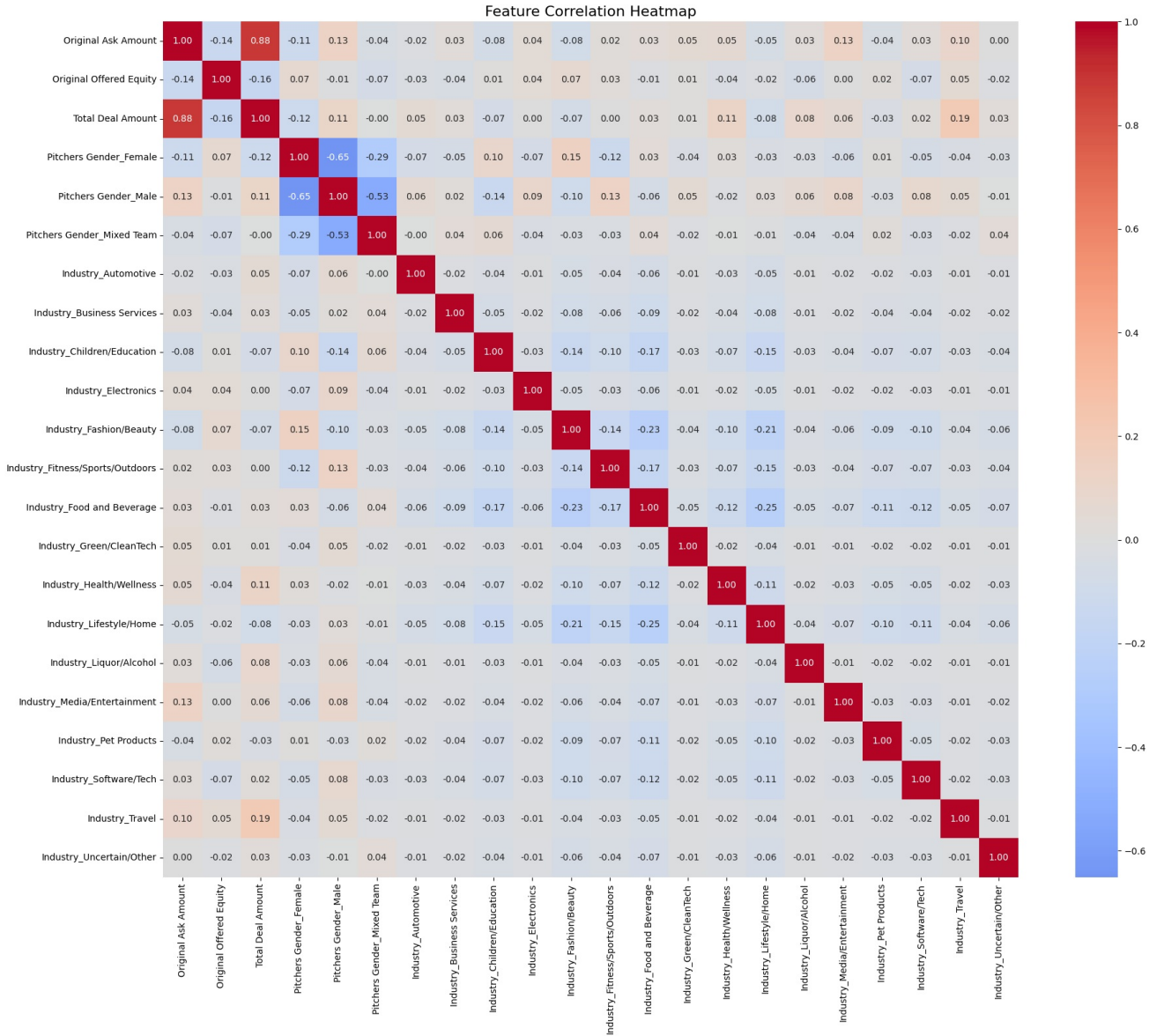
Lori Greiner Present

Robert Herjavec Present

Daymond John Present

Kevin O'Leary Present

Guest Present: Whether any guest shark was present; could influence deal dynamics.



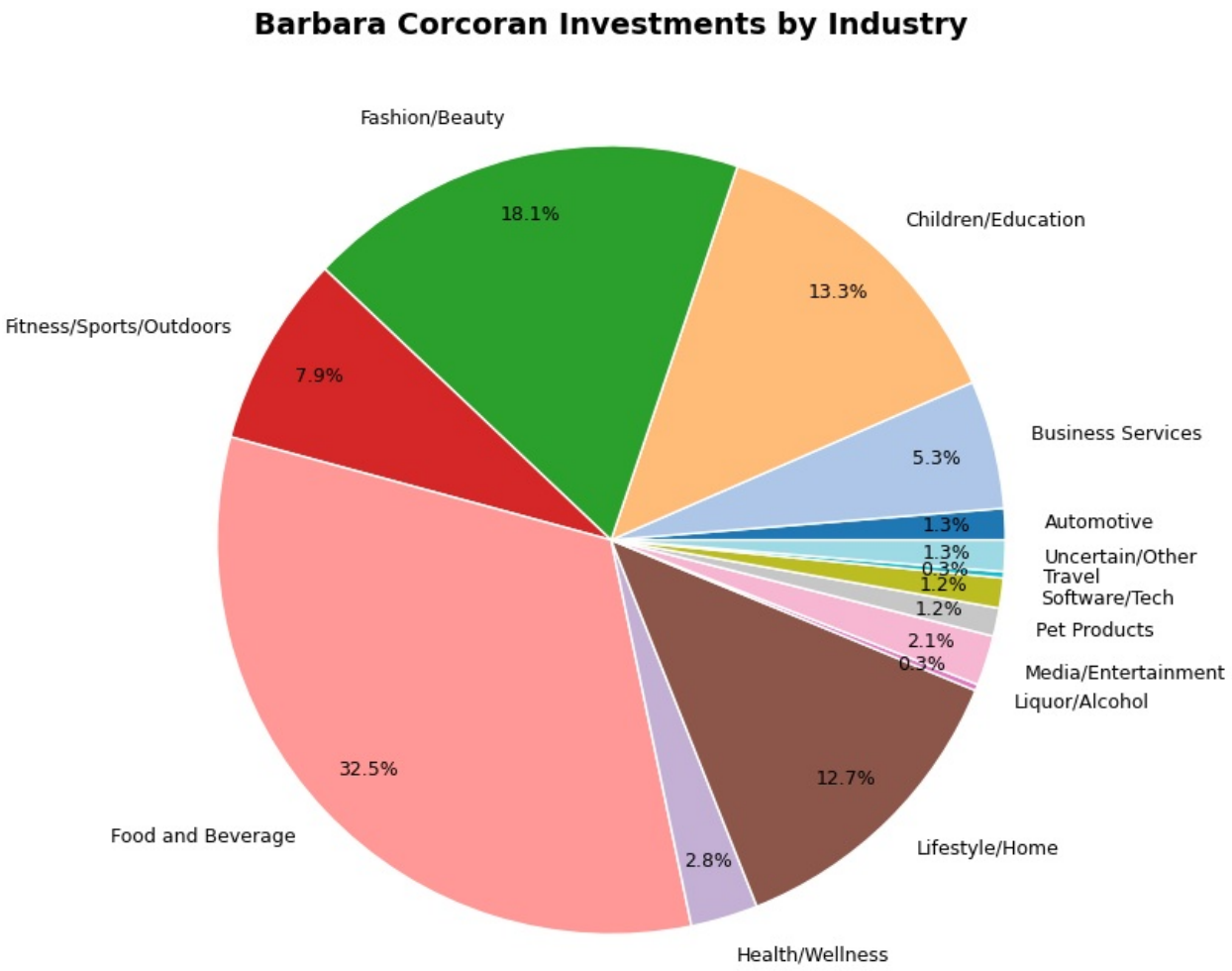

```
Season Number: [1 1]
Startup Name: ['AvaTheElephant' "MrTod'sPieFactory"]
Episode Number: [1 1]
Pitch Number: [1 2]
Season Start: ['9-Aug-09' '9-Aug-09']
Season End: ['5-Feb-10' '5-Feb-10']
Original Air Date: ['9-Aug-09' '9-Aug-09']
```

```
Industry: ['Health/Wellness' 'Food and Beverage']
Business Description: ['Ava The Elephant - Baby and Child Care'
 "Mr. Tod's Pie Factory - Specialty Food"]
Company Website: ['http://www.avatheelephant.com/' 'http://whybake.com/']
Pitchers Gender: ['Female' 'Male']
Pitchers Average Age: [nan nan]
Pitchers City: ['Atlanta' 'Somerset']
Pitchers State: ['GA' 'NJ']
Entrepreneur Names: ['Tiffany Krumins' 'Tod Wilson']
Multiple Entrepreneurs: [0. 0.]
US Viewership: [4.15 4.15]
Original Ask Amount: [ 50000. 460000.]
Original Offered Equity: [15. 10.]
Valuation Requested: [ 333333. 4600000.]
Got Deal: [1 1]
Total Deal Amount: [ 50000. 460000.]
Total Deal Equity: [55. 50.]
Deal Valuation: [ 90909. 920000.]
Number of Sharks in Deal: [1. 2.]
Investment Amount Per Shark: [ 50000. 230000.]
Equity Per Shark: [55. 25.]
Royalty Deal: [nan nan]
Advisory Shares Equity: [nan nan]
Loan: [nan nan]
Deal Has Conditions: [nan nan]
Barbara Corcoran Investment Amount: [ 50000. 230000.]
Barbara Corcoran Investment Equity: [55. 25.]
Mark Cuban Investment Amount: [nan nan]
Mark Cuban Investment Equity: [nan nan]
Lori Greiner Investment Amount: [nan nan]
Lori Greiner Investment Equity: [nan nan]
Robert Herjavec Investment Amount: [nan nan]
Robert Herjavec Investment Equity: [nan nan]
Daymond John Investment Amount: [    nan 230000.]
Daymond John Investment Equity: [nan 25.]
Kevin O Leary Investment Amount: [nan nan]
Kevin O Leary Investment Equity: [nan nan]
Guest Investment Amount: [nan nan]
Guest Investment Equity: [nan nan]
Guest Name: [nan nan]
Barbara Corcoran Present: [1. 1.]
Mark Cuban Present: [0. 0.]
Lori Greiner Present: [0. 0.]
Robert Herjavec Present: [1. 1.]
Daymond John Present: [1. 1.]
Kevin O Leary Present: [1. 1.]
Guest Present: [nan nan]
```

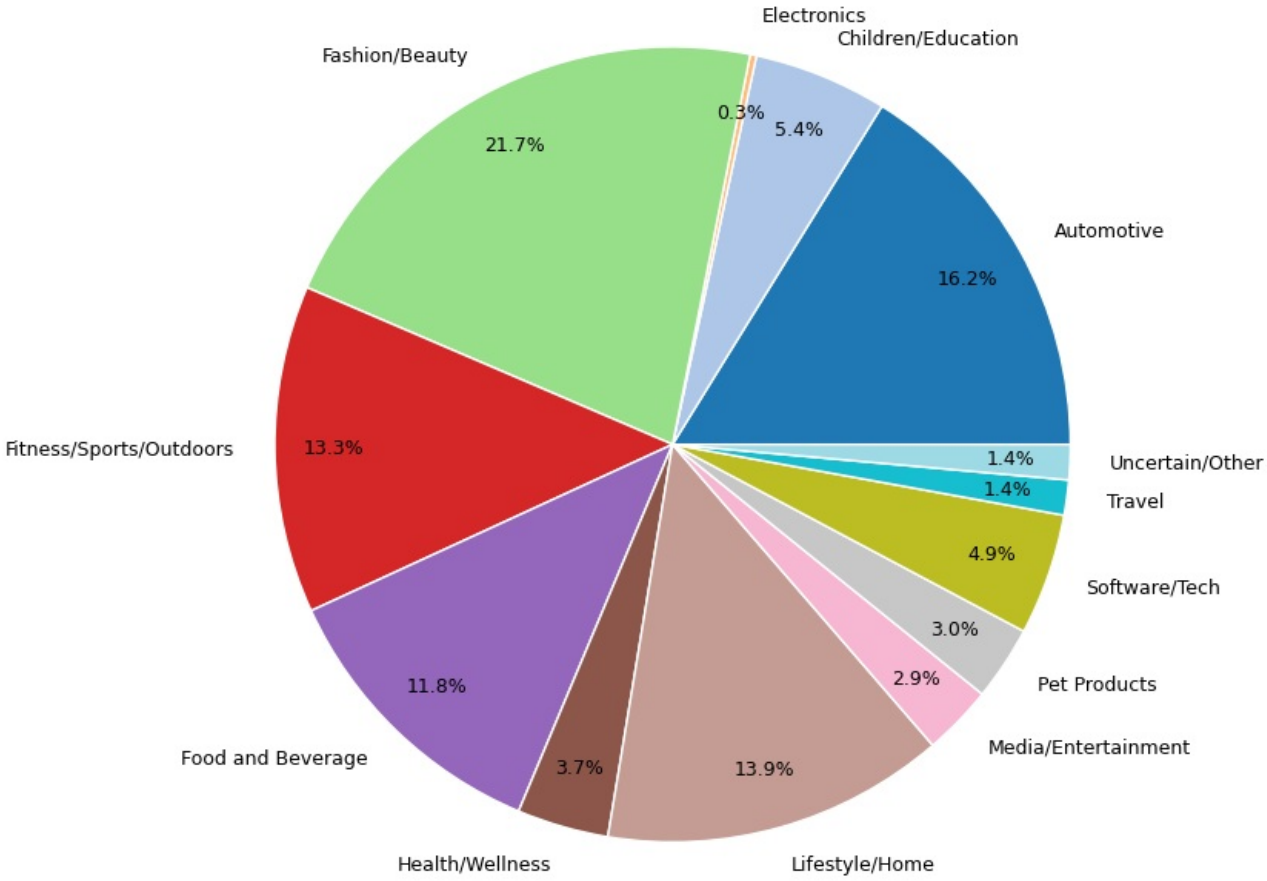
In [ ]:

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

def create_individual_shark_pie_charts(df):
    sharks = ['Barbara Corcoran', 'Daymond John', 'Kevin O Leary', 'Robert Herjavec', 'Mark Cuban', 'Lori Greine
    investment_cols = [f'{shark} Investment Amount' for shark in sharks]

    def create_shark_pie_chart(shark, investment_col):
        shark_investments = df[['Industry', investment_col]].copy()
        shark_investments = shark_investments[shark_investments[investment_col].notna()]

        if shark_investments.empty:
            print(f"No investment data for {shark}")
            return None
        industry_investments = shark_investments.groupby('Industry')[investment_col].sum()
        total_investment = industry_investments.sum()
        num_slices = len(industry_investments)
        colors = plt.cm.tab20(np.linspace(0, 1, num_slices))
        plt.figure(figsize=(10, 7))
        def make_autopct(values):
            def my_autopct(pct):
                return f'{pct:.1f}%'
            return my_autopct

        plt.pie(
            industry_investments,
            labels=industry_investments.index,
            colors=colors,
            autopct=make_autopct(industry_investments),
            pctdistance=0.85,
            labeldistance=1.1,
```

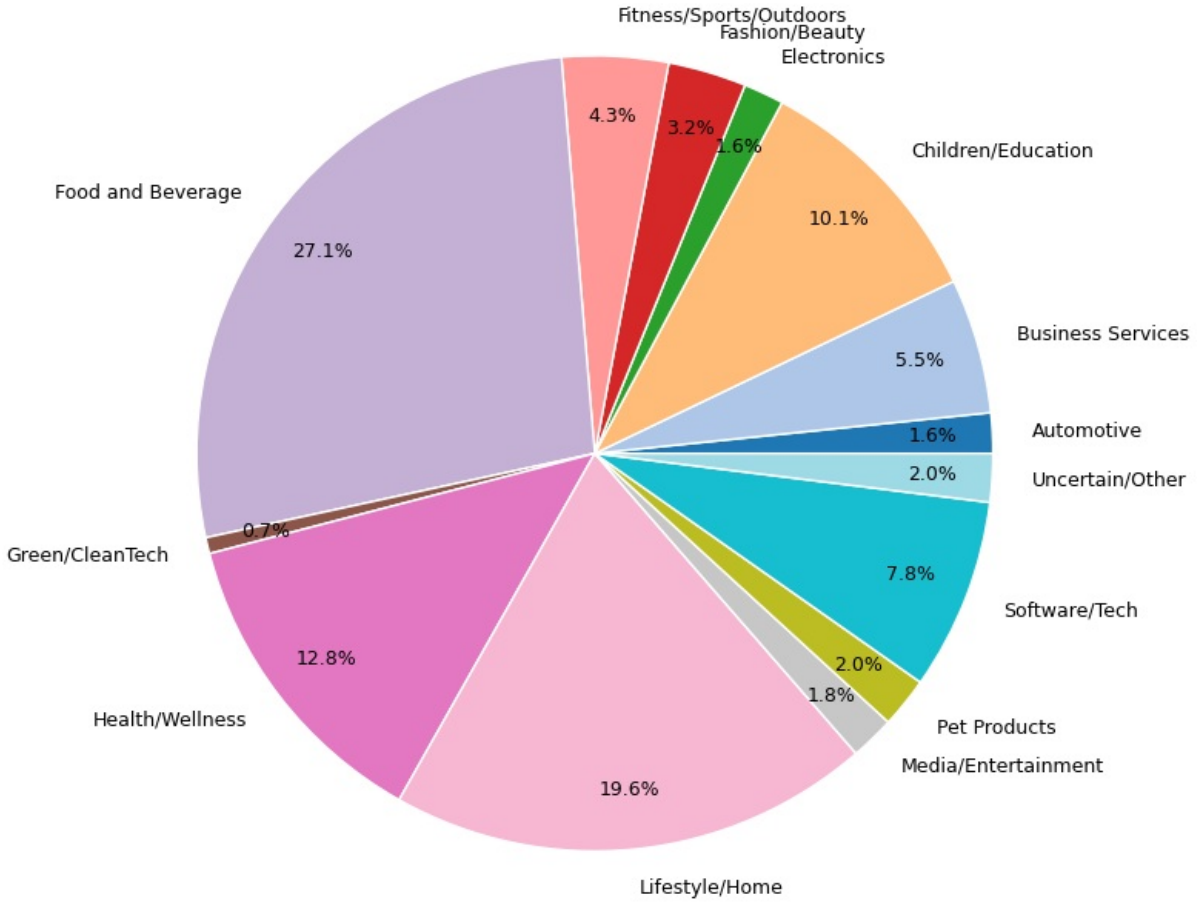
```
            wedgeprops={'edgecolor': 'white', 'linewidth': 1},
            textprops={'fontsize': 9}
        )

        plt.title(f'{shark} Investments by Industry', fontsize=14, fontweight='bold')
        plt.tight_layout()
        plt.savefig(f'{shark.replace(" ", "_")}_investments_pie_chart.png', bbox_inches='tight')
        plt.close()

        return industry_investments

    shark_investments = {}
    for shark, col in zip(sharks, investment_cols):
        investments = create_shark_pie_chart(shark, col)
        if investments is not None:
            shark_investments[shark] = investments

    print("\nDetailed Shark Investments by Industry:")
    for shark, investments in shark_investments.items():
        print(f"\n{shark}:")
        print(investments)

create_individual_shark_pie_charts(df)
```



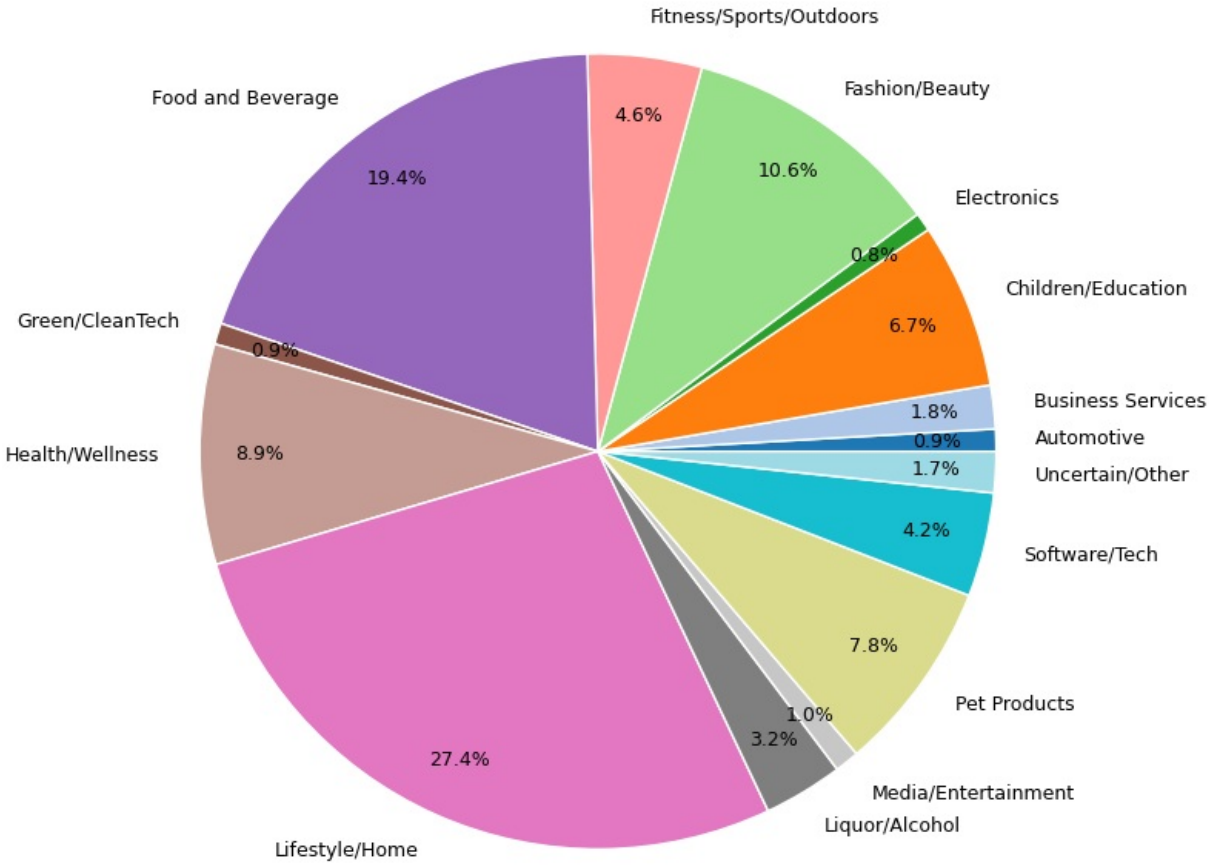
## Daymond John Investments by Industry



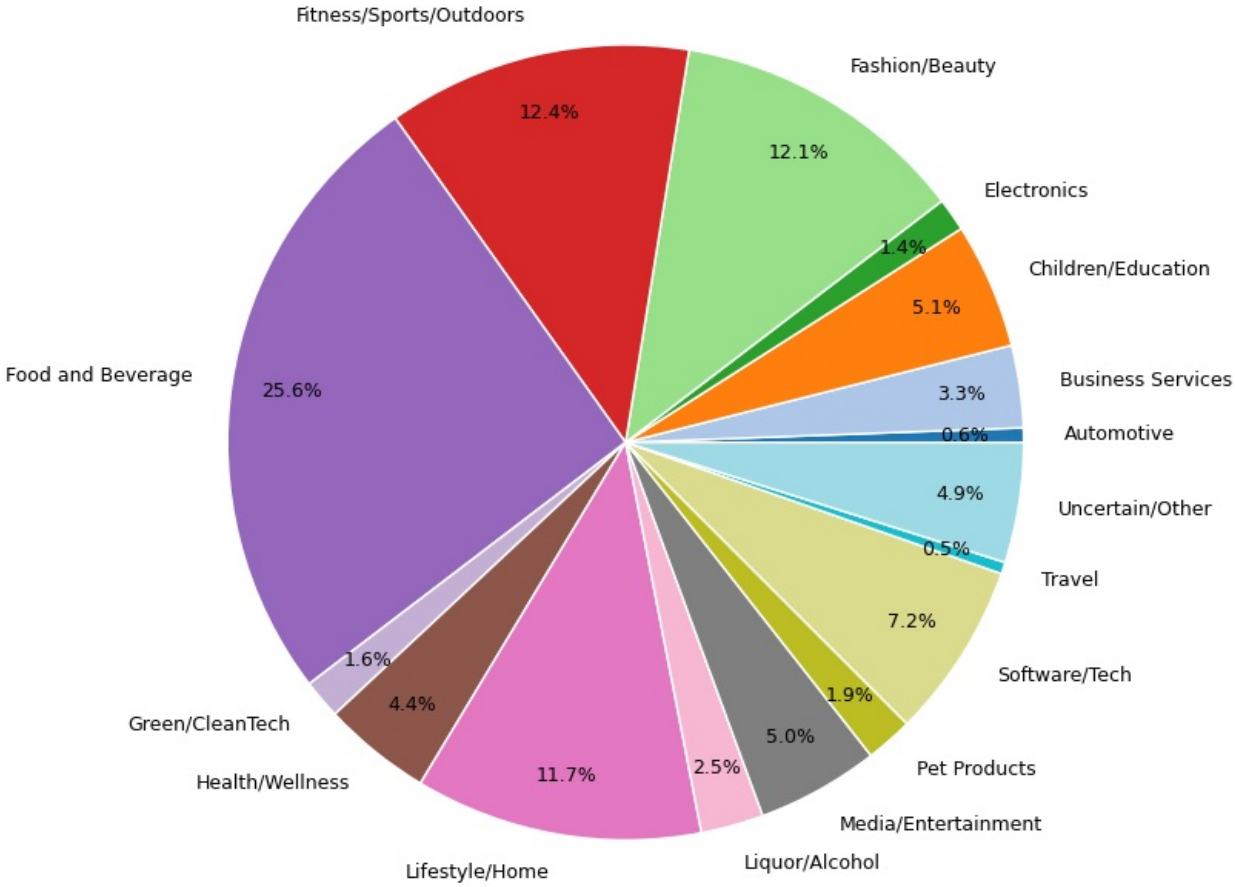

## Kevin O Leary Investments by Industry

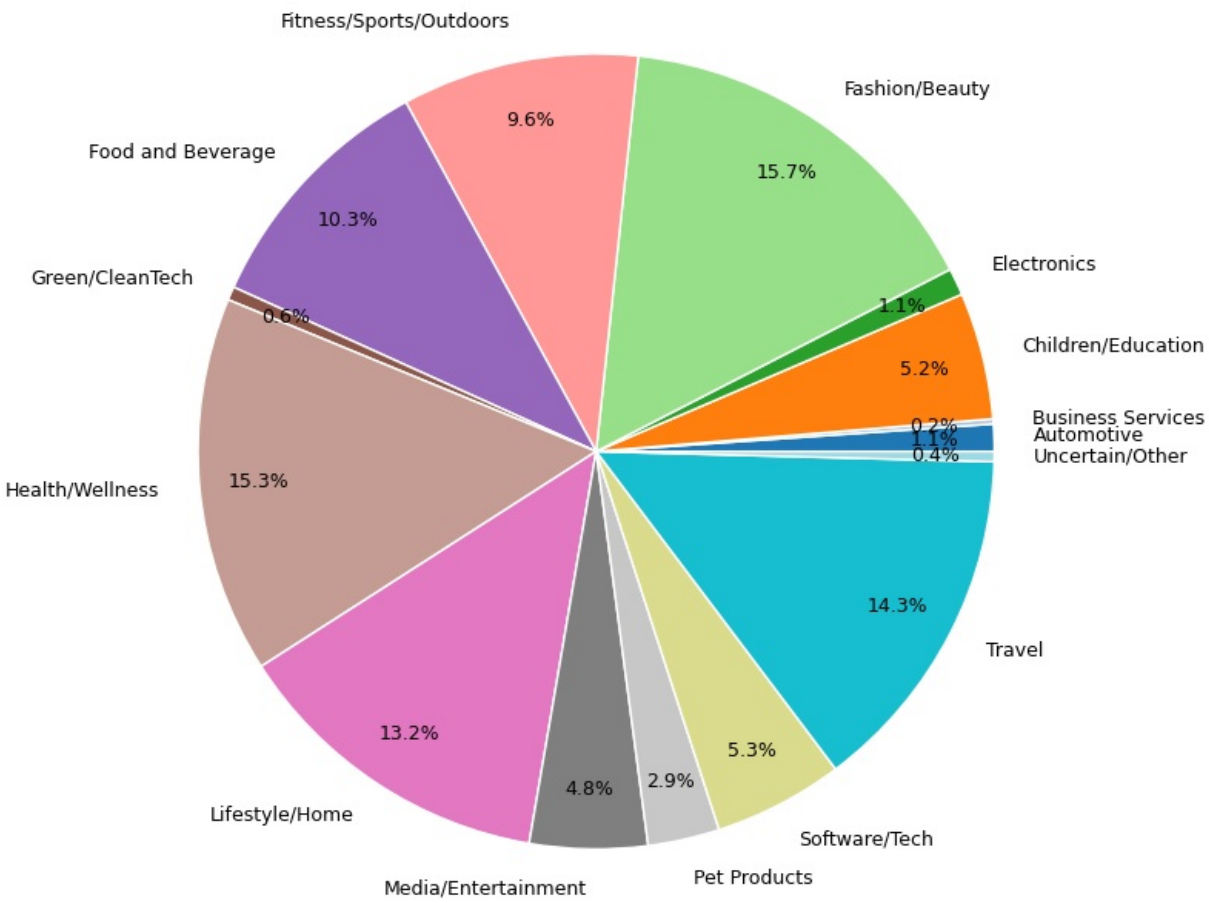


## Lori Greiner Investments by Industry




## Mark Cuban Investments by Industry






In [ ]:
```
total_nan_count = df.isna().sum().sum()
print("Total number of NaN values in the DataFrame:", total_nan_count)
```

```
Total number of NaN values in the DataFrame: 34467
```

In [ ]:
```
columns_to_drop = [
    'Season Start', 'Season End', 'Original Air Date', 'Pitch Number',
    'Company Website', 'Pitchers Average Age',
    'Total Deal Amount', 'Total Deal Equity', 'Deal Valuation',
    'Royalty Deal', 'Advisory Shares Equity', 'Loan', 'Deal Has Conditions', 'Barbara Corcoran Investment Equity
    'Mark Cuban Investment Equity','Lori Greiner Investment Equity','Robert Herjavec Investment Equity','Daymon
    'Kevin O Leary Investment Equity',
    'Guest Investment Amount', 'Guest Investment Equity', 'Guest Name',
    'Guest Present', 'Got Deal', 'Number of Sharks in Deal', 'Investment Amount Per Shark',
    'Equity Per Shark', 'Episode Number', 'Season Number',  'Entrepreneur Names'
]

df = df.drop(columns=columns_to_drop)
print(df.columns)
```

We are dropping the above columns and the reasons are,

Season Start, Season End, Original Air Date: Temporal information irrelevant to prediction.

Pitch Number, Episode Number, Season Number: Episode-specific details not influencing investments.

Entrepreneur Names: Non-predictive and irrelevant to shark decisions.

Company Website: URL data not used for prediction unless analyzing content.

Pitchers Average Age: Potentially less impactful than other demographic or team features.

Total Deal Amount, Total Deal Equity, Deal Valuation: Outcome variables causing data leakage.

Got Deal: Target outcome for other predictions, not for individual shark investments.

Barbara Corcoran Investment Equity, Mark Cuban Investment Equity, etc.: Redundant for modeling investment amounts.

Guest Investment Amount, Guest Investment Equity, Guest Name, Guest Present: Guest-related columns not consistently informative.

Royalty Deal, Advisory Shares Equity, Loan, Deal Has Conditions: Deal structuring details not directly predictive.

Number of Sharks in Deal: Redundant if individual investments are included.

Investment Amount Per Shark, Equity Per Shark: Derived metrics not needed for prediction.

Selected column names

```
Index(['Startup Name', 'Industry', 'Business Description', 'Pitchers Gender',
       'Pitchers City', 'Pitchers State', 'Multiple Entrepreneurs',
       'US Viewership', 'Original Ask Amount', 'Original Offered Equity',
       'Valuation Requested', 'Barbara Corcoran Investment Amount',
       'Mark Cuban Investment Amount', 'Lori Greiner Investment Amount',
       'Robert Herjavec Investment Amount', 'Daymond John Investment Amount',
       'Kevin O Leary Investment Amount', 'Barbara Corcoran Present',
       'Mark Cuban Present', 'Lori Greiner Present', 'Robert Herjavec Present',
       'Daymond John Present', 'Kevin O Leary Present'],
      dtype='object')
```

In [ ]:
```
for col in df.columns:
    print(f"{col}: {df[col].iloc[0:6].values}")
```

Now we print some values for each of the selected columns

```
Startup Name: ['AvaTheElephant' "MrTod'sPieFactory" 'Wispots' 'CollegeFoxesPackingBoxes'
 'IonicEar' 'APerfectPear']
Industry: ['Health/Wellness' 'Food and Beverage' 'Business Services'
 'Lifestyle/Home' 'Software/Tech' 'Food and Beverage']
Business Description: ['Ava The Elephant - Baby and Child Care'
 "Mr. Tod's Pie Factory - Specialty Food" 'Wispots - Consumer Services'
 'College Foxes Packing Boxes - Consumer Services' 'Ionic Ear - Novelties'
 'A Perfect Pear - Specialty Food']
Pitchers Gender: ['Female' 'Male' 'Male' 'Male' 'Male' 'Female']
Pitchers City: ['Atlanta' 'Somerset' 'Cary' 'Tampa' 'St. Paul' 'Napa Valley']
Pitchers State: ['GA' 'NJ' 'NC' 'FL' 'MN' 'CA']
Multiple Entrepreneurs: [0. 0. 0. 0. 0. 0.]
US Viewership: [4.15 4.15 4.15 4.15 4.15 5.59]
Original Ask Amount: [  50000.  460000. 1200000.  250000. 1000000.  500000.]
Original Offered Equity: [15. 10. 10. 25. 15. 15.]
Valuation Requested: [  333333.  4600000. 12000000.  1000000.  6666667.  3333333.]
Barbara Corcoran Investment Amount: [ 50000. 230000.     nan     nan     nan     nan]
Mark Cuban Investment Amount: [nan nan nan nan nan nan]
Lori Greiner Investment Amount: [nan nan nan nan nan nan]
Robert Herjavec Investment Amount: [    nan     nan     nan     nan     nan 250000.]
Daymond John Investment Amount: [    nan 230000.     nan     nan     nan     nan]
Kevin O Leary Investment Amount: [    nan     nan     nan     nan     nan 250000.]
Barbara Corcoran Present: [1. 1. 1. 1. 1. 1.]
Mark Cuban Present: [0. 0. 0. 0. 0. 0.]
Lori Greiner Present: [0. 0. 0. 0. 0. 0.]
Robert Herjavec Present: [1. 1. 1. 1. 1. 1.]
Daymond John Present: [1. 1. 1. 1. 1. 1.]
Kevin O Leary Present: [1. 1. 1. 1. 1. 1.]
```

In [ ]:
```
df['Sharks Invested'] = df[['Barbara Corcoran Investment Amount',
                            'Mark Cuban Investment Amount',
                            'Lori Greiner Investment Amount',
                            'Robert Herjavec Investment Amount',
                            'Daymond John Investment Amount',
                            'Kevin O Leary Investment Amount']].notna().astype(int).values.tolist()
df['Sharks Investment Amounts'] = df[['Barbara Corcoran Investment Amount',
                                      'Mark Cuban Investment Amount',
                                      'Lori Greiner Investment Amount',
                                      'Robert Herjavec Investment Amount',
                                      'Daymond John Investment Amount',
                                      'Kevin O Leary Investment Amount']].values.tolist()
df['Sharks Investment Amounts'] = df['Sharks Investment Amounts'].apply(
    lambda x: [0 if pd.isna(val) else val for val in x]
)
columns_to_drop = [
    'Barbara Corcoran Investment Amount', 'Mark Cuban Investment Amount',
    'Lori Greiner Investment Amount', 'Robert Herjavec Investment Amount',
    'Daymond John Investment Amount', 'Kevin O Leary Investment Amount'
]
df = df.drop(columns=columns_to_drop)
```

In the above cell, we combine investment data for all sharks into two compact columns:

Sharks Invested: Binary indicators showing which sharks invested. Sharks Investment Amounts: The actual investment amounts (with 0 for no investment).

Simplifies the dataset by removing unnecessary individual columns.

Makes the data more suitable for machine learning models that can use these compact representations.

```
Index 0: Barbara Corcoran
Index 1: Mark Cuban
Index 2: Lori Greiner
Index 3: Robert Herjavec
Index 4: Daymond John
Index 5: Kevin O'Leary
```

In [ ]:
```
total_nan_count = df.isna().sum().sum()
print("Total number of NaN values in the DataFrame:", total_nan_count)
```

```
Total number of NaN values in the DataFrame: 4238
```

In [ ]:
```
nan_counts = df.isna().sum()
print("NaN counts per column:")
for col, count in nan_counts[nan_counts > 0].items():
    print(f"{col}: {count} NaN values")
```

```
NaN counts per column:
Pitchers Gender: 7 NaN values
Pitchers City: 834 NaN values
Pitchers State: 566 NaN values
Multiple Entrepreneurs: 427 NaN values
US Viewership: 4 NaN values
Original Ask Amount: 1 NaN values
Original Offered Equity: 1 NaN values
Valuation Requested: 1 NaN values
Barbara Corcoran Present: 423 NaN values
Mark Cuban Present: 373 NaN values
Lori Greiner Present: 373 NaN values
Robert Herjavec Present: 429 NaN values
Daymond John Present: 423 NaN values
Kevin O Leary Present: 376 NaN values
```

In [ ]:
```
df['Multiple Entrepreneurs'] = df['Multiple Entrepreneurs'].fillna(0)
df['US Viewership'] = df['US Viewership'].fillna(df['US Viewership'].median())
numeric_cols = ['Original Ask Amount', 'Original Offered Equity', 'Valuation Requested']
for col in numeric_cols:
    df[col] = df[col].fillna(df[col].median())
```

In [ ]:
```
present_columns = [col for col in df.columns if 'Present' in col]
for col in present_columns:
    df[col] = df[col].fillna(0)
```

In [ ]:
```
total_rows = len(df)
nan_counts = df.isna().sum()
print("NaN value analysis per column:")
for col, count in nan_counts[nan_counts > 0].items():
    percentage = (count / total_rows) * 100
    print(f"{col}: {count} NaN values out of {total_rows} total rows ({percentage:.1f}%)")
```

In [ ]:
```
df = df.drop(['Pitchers City', 'Pitchers State'], axis=1)
df['Pitchers Gender'] = df['Pitchers Gender'].fillna(0)
nan_counts = df.isna().sum()
print("NaN value analysis after changes:")
for col, count in nan_counts[nan_counts > 0].items():
    percentage = (count / total_rows) * 100
    if count > 0:
        print(f"{col}: {count} NaN values out of {total_rows} total rows ({percentage:.1f}%)")
```

In [ ]:
```

```

```
NaN counts per column:
```

In [ ]:
```
for col in df.columns:
```

```
    print(f"{col}: {df[col].iloc[0:6].values}")
```

```
Startup Name: ['AvaTheElephant' "MrTod'sPieFactory" 'Wispots' 'CollegeFoxesPackingBoxes'
 'IonicEar' 'APerfectPear']
Industry: ['Health/Wellness' 'Food and Beverage' 'Business Services'
 'Lifestyle/Home' 'Software/Tech' 'Food and Beverage']
Business Description: ['Ava The Elephant - Baby and Child Care'
 "Mr. Tod's Pie Factory - Specialty Food" 'Wispots - Consumer Services'
 'College Foxes Packing Boxes - Consumer Services' 'Ionic Ear - Novelties'
 'A Perfect Pear - Specialty Food']
Pitchers Gender: ['Female' 'Male' 'Male' 'Male' 'Male' 'Female']
Multiple Entrepreneurs: [0. 0. 0. 0. 0. 0.]
US Viewership: [4.15 4.15 4.15 4.15 4.15 5.59]
Original Ask Amount: [  50000.  460000. 1200000.  250000. 1000000.  500000.]
Original Offered Equity: [15. 10. 10. 25. 15. 15.]
Valuation Requested: [  333333.  4600000. 12000000.  1000000.  6666667.  3333333.]
Barbara Corcoran Present: [1. 1. 1. 1. 1. 1.]
Mark Cuban Present: [0. 0. 0. 0. 0. 0.]
Lori Greiner Present: [0. 0. 0. 0. 0. 0.]
Robert Herjavec Present: [1. 1. 1. 1. 1. 1.]
Daymond John Present: [1. 1. 1. 1. 1. 1.]
Kevin O Leary Present: [1. 1. 1. 1. 1. 1.]
Sharks Invested: [list([1, 0, 0, 0, 0, 0]) list([1, 0, 0, 0, 1, 0])
 list([0, 0, 0, 0, 0, 0]) list([0, 0, 0, 0, 0, 0])
 list([0, 0, 0, 0, 0, 0]) list([0, 0, 0, 1, 0, 1])]
Sharks Investment Amounts: [list([50000.0, 0, 0, 0, 0, 0]) list([230000.0, 0, 0, 0, 230000.0,
0])
 list([0, 0, 0, 0, 0, 0]) list([0, 0, 0, 0, 0, 0])
 list([0, 0, 0, 0, 0, 0]) list([0, 0, 0, 250000.0, 0, 250000.0])]
```

# Generating and Saving Text Embeddings for Selected Columns

This code uses OpenAI's embedding model to transform text data from specific columns (columns_to_embed) into numerical embeddings. For each column, it computes embeddings for every row, saves the results as .npy files, and organizes them into a dictionary (embeddings_dict) for efficient storage and retrieval.

In [ ]:
```python
import openai
import numpy as np
import os
from openai import OpenAI
client = OpenAI(
    api_key=,
)

columns_to_embed = ['Startup Name', 'Industry', 'Business Description', 'Pitchers Gender']
MODEL_NAME = "text-embedding-3-small"

def get_embedding(text, model=MODEL_NAME):
    response = client.embeddings.create(input=[text.replace("\n", " ")], model=model)
    return response.data[0].embedding
embeddings_dict = {}
for column in columns_to_embed:
    embeddings = [get_embedding(str(text)) for text in df[column]]
    embeddings_array = np.array(embeddings)
    np.save(f"{column}_embeddings.npy", embeddings_array)
    embeddings_dict[column] = embeddings_array
```

The reason for converting the column values into embeddings is to capture the semantic and contextual information of the text data present in columns_to_embed. For instance, embeddings for columns like Industry and Business Description can encode nuanced relationships and align with demographic attributes such as Pitchers Gender, potentially revealing patterns that influence investment decisions. Similarly, Pitchers City and Pitchers State embeddings can capture geographic trends, such as industries thriving in specific locations, making startups in those areas stronger candidates for investment. This approach allows the model to leverage complex, latent relationships in the data, which might serve as significant predictors for investments.

This code visualizes text embeddings from selected columns of a dataset ( `Startup Name` , `Industry` , etc.) in a 3D space to explore semantic relationships between terms. It computes the average embeddings for unique terms in each column, selects the most frequent terms (up to a specified limit), and reduces the high-dimensional embeddings to three dimensions using t-SNE. The reduced embeddings are then plotted interactively using Plotly, where points represent terms, and their size corresponds to their frequency in the dataset. This visualization helps in understanding patterns and similarities across categories, making the data easier to interpret and analyze.

In [ ]:
```python
import numpy as np
import plotly.express as px
from sklearn.manifold import TSNE
from collections import defaultdict
```

```python
import pandas as pd

def create_3d_combined_embeddings(df, embeddings_dict, columns, max_points_per_category=20):
    all_embeddings = []
    all_labels = []
    all_categories = []
    all_frequencies = []
    for column in columns:
        unique_items = defaultdict(list)
        for idx, item in enumerate(df[column]):
            unique_items[str(item)].append(embeddings_dict[column][idx])
        items_freq = [(item, len(emb_list)) for item, emb_list in unique_items.items()]
        items_freq.sort(key=lambda x: x[1], reverse=True)
        selected_items = items_freq[:max_points_per_category]
        for item, freq in selected_items:
            emb_list = unique_items[item]
            all_embeddings.append(np.mean(emb_list, axis=0))
            all_labels.append(item)
            all_categories.append(column)
            all_frequencies.append(freq)

    embeddings_array = np.array(all_embeddings)
    tsne = TSNE(n_components=3, perplexity=min(30, len(all_labels)-1),
                random_state=42)
    reduced_embeddings = tsne.fit_transform(embeddings_array)
    plot_df = pd.DataFrame({
        'x': reduced_embeddings[:, 0],
        'y': reduced_embeddings[:, 1],
        'z': reduced_embeddings[:, 2],
        'word': all_labels,
        'category': all_categories,
        'frequency': all_frequencies
    })
    fig = px.scatter_3d(
        plot_df,
        x='x',
        y='y',
        z='z',
        color='category',
        text='word',
        size='frequency',
        hover_data={
            'x': False,
            'y': False,
            'z': False,
            'word': True,
            'category': True,
            'frequency': True
        },
        title='3D Word Embeddings Visualization (Top Terms)',
        template='plotly_white',
        color_discrete_sequence=px.colors.qualitative.Set3
    )

    fig.update_traces(
        textposition='top center',
        marker=dict(opacity=0.8),
        textfont=dict(size=12),
        hovertemplate="<br>".join([
            "<b>%{text}</b>",
            "Category: %{customdata[1]}",
            "Frequency: %{customdata[2]}"
        ])
    )

    fig.update_layout(
        scene=dict(
            xaxis_title='',
            yaxis_title='',
            zaxis_title='',
            xaxis=dict(showgrid=False, showticklabels=False),
            yaxis=dict(showgrid=False, showticklabels=False),
            zaxis=dict(showgrid=False, showticklabels=False),
            bgcolor='white'
        ),
        hoverlabel=dict(bgcolor="white"),
        width=1200,
        height=800,
        showlegend=True,
        legend_title_text='Categories',
        legend=dict(
            yanchor="top",
            y=0.99,
```

```
            xanchor="left",
            x=0.01
        ),
        title=dict(
            text='3D Word Embeddings Visualization (Top Terms)',
            y=0.95
        )
    )

    return fig
columns_to_embed = ['Startup Name', 'Industry', 'Business Description', 'Pitchers Gender']
embeddings_dict = {}
for column in columns_to_embed:
    embeddings_dict[column] = np.load(f"{column}_embeddings.npy")
fig = create_3d_combined_embeddings(df, embeddings_dict, columns_to_embed, max_points_per_category=150)
fig.show()
```

In [ ]:
```
fig.write_html("word_embeddings_3d.html")
```

In [ ]:
```

```

Now load the embedding files to be used for training and testing

In [ ]:
```
import numpy as np
import pandas as pd
columns_to_replace = ['Startup Name', 'Industry', 'Business Description', 'Pitchers Gender']
for column in columns_to_replace:
    embeddings = np.load(f"Startup Name_embeddings.npy")
    assert embeddings.shape[0] == df.shape[0], f"Row count mismatch for column {column}!"
    df[column] = list(embeddings)
```

In [ ]:
```
nan_counts = df.isna().sum()
print("NaN counts per column:")
for col, count in nan_counts[nan_counts > 0].items():
    print(f"{col}: {count} NaN values")
```

In [ ]:
```
total_nan_count = df.isna().sum().sum()
print("Total number of NaN values in the DataFrame:", total_nan_count)
```

In [ ]:
```
for column in df.columns:
    print(f"{column}: {df[column].iloc[0]}")
```

```
Startup Name: [ 0.04440477 -0.05706164 -0.00765378 ...  0.00296032 -0.01283056
  0.0243169 ]
Industry: [ 0.04440477 -0.05706164 -0.00765378 ...  0.00296032 -0.01283056
  0.0243169 ]
Business Description: [ 0.04440477 -0.05706164 -0.00765378 ...  0.00296032 -0.01283056
  0.0243169 ]
Pitchers Gender: [ 0.04440477 -0.05706164 -0.00765378 ...  0.00296032 -0.01283056
  0.0243169 ]
Multiple Entrepreneurs: 0.0
US Viewership: 4.15
Original Ask Amount: 50000.0
Original Offered Equity: 15.0
Valuation Requested: 333333.0
Barbara Corcoran Present: 1.0
Mark Cuban Present: 0.0
Lori Greiner Present: 0.0
Robert Herjavec Present: 1.0
Daymond John Present: 1.0
Kevin O Leary Present: 1.0
Sharks Invested: [1, 0, 0, 0, 0, 0]
Sharks Investment Amounts: [50000.0, 0, 0, 0, 0, 0]
```

This code reduces high-dimensional text embeddings for columns like Startup Name and Industry into a single numerical value using PCA, capturing the most significant variance in the data. This simplifies the embeddings for easier integration into machine learning models while retaining essential semantic and contextual information.

In [ ]:
```
from sklearn.decomposition import PCA

pca = PCA(n_components=1)
for column in ['Startup Name', 'Industry', 'Business Description', 'Pitchers Gender']:
    embeddings_matrix = np.stack(df[column].values)
    reduced_embeddings = pca.fit_transform(embeddings_matrix)
    df[column] = reduced_embeddings.flatten()
```

Use MinMaxScaler to normalize all column values

In [ ]:
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
df[df.columns[:-2]] = scaler.fit_transform(df[df.columns[:-2]])
print(df.head())
```

```
   Startup Name  Industry  Business Description  Pitchers Gender  \
0      0.360928  0.360967              0.360955         0.360830
1      0.133302  0.133300              0.133316         0.133521
2      0.705327  0.705066              0.705400         0.704915
3      0.427710  0.427480              0.427781         0.427869
4      0.880520  0.880574              0.880583         0.879797

   Multiple Entrepreneurs  US Viewership  Original Ask Amount  \
0                     0.0       0.295133             0.008016
1                     0.0       0.295133             0.090180
2                     0.0       0.295133             0.238477
3                     0.0       0.295133             0.048096
4                     0.0       0.295133             0.198397

   Original Offered Equity  Valuation Requested  Barbara Corcoran Present  \
0                 0.141414             0.002935                       1.0
1                 0.090909             0.045618                       1.0
2                 0.090909             0.119648                       1.0
3                 0.242424             0.009604                       1.0
4                 0.141414             0.066293                       1.0

   Mark Cuban Present  Lori Greiner Present  Robert Herjavec Present  \
0                 0.0                   0.0                      1.0
1                 0.0                   0.0                      1.0
2                 0.0                   0.0                      1.0
3                 0.0                   0.0                      1.0
4                 0.0                   0.0                      1.0

   Daymond John Present  Kevin O Leary Present     Sharks Invested  \
0                   1.0                   1.0  [1, 0, 0, 0, 0, 0]
1                   1.0                   1.0  [1, 0, 0, 0, 1, 0]
2                   1.0                   1.0  [0, 0, 0, 0, 0, 0]
3                   1.0                   1.0  [0, 0, 0, 0, 0, 0]
4                   1.0                   1.0  [0, 0, 0, 0, 0, 0]

          Sharks Investment Amounts
0          [50000.0, 0, 0, 0, 0, 0]
1  [230000.0, 0, 0, 0, 230000.0, 0]
2                [0, 0, 0, 0, 0, 0]
3                [0, 0, 0, 0, 0, 0]
4                [0, 0, 0, 0, 0, 0]
```

In [ ]:
```
df.columns.to_list()
```

```
['Startup Name',
 'Industry',
 'Business Description',
 'Pitchers Gender',
 'Multiple Entrepreneurs',
 'US Viewership',
 'Original Ask Amount',
 'Original Offered Equity',
 'Valuation Requested',
 'Barbara Corcoran Present',
 'Mark Cuban Present',
 'Lori Greiner Present',
 'Robert Herjavec Present',
 'Daymond John Present',
 'Kevin O Leary Present',
 'Sharks Invested',
 'Sharks Investment Amounts']
```

In [ ]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from xgboost import XGBClassifier
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from imblearn.over_sampling import SMOTE
import warnings
warnings.filterwarnings('ignore')
```

We create a copy of df to be used for regression task.

Now we consider 4 classification models for predicting which sharks are gonna invest. These models are RandomForestClassifier, LogisticRegression, SVM and XGBClassifier. These four models were chosen to cover a range of strengths and approaches for classification. Random Forest is robust, captures non-linear relationships, and helps understand feature importance. Logistic Regression provides a simple, interpretable baseline for comparison. SVM is effective for non-linear decision boundaries and works well with standardized data. XGBoost, a powerful gradient-boosting model, excels at capturing complex patterns and is widely used for its performance. This mix ensures a balanced evaluation of different algorithms for predicting shark investments.

In [ ]:
```
df_regres = df.copy()
y = np.array([np.array(x) for x in df['Sharks Invested']])
X = df.drop(columns=['Sharks Invested', 'Sharks Investment Amounts'])
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

In [ ]:
```
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_state=42)
X_valid, X_test, y_valid, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)
```

How SMOTE Works:

Identify Minority Class Instances: SMOTE first identifies the instances that belong to the minority class.

Find Nearest Neighbors: For each minority-class instance, SMOTE finds a specified number of its closest neighbors within the minority class.

Generate Synthetic Samples: Instead of merely duplicating minority samples, SMOTE creates new, synthetic examples by interpolating between the selected instance and its neighbors. Essentially, it picks a point between the two existing samples to form a new, more diverse example of the minority class.

In [ ]:
```
def apply_smote(X_train, y_shark, random_state=42):
    unique = np.unique(y_shark)
    if len(unique) > 1:
        smote = SMOTE(random_state=random_state)
        X_train_balanced, y_train_balanced = smote.fit_resample(X_train, y_shark)
        print("Applied SMOTE balancing")
    else:
        X_train_balanced, y_train_balanced = X_train, y_shark
        print("Skipping SMOTE - only one class present")
    return X_train_balanced, y_train_balanced
```

In [ ]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, classification_report

def tune_hyperparameters(X_train, X_valid, y_train, y_valid, models, param_grids):
    tuned_models = {}
    for model_name, model in models.items():
        print(f"\nTuning {model_name}")
        grid_search = GridSearchCV(
            estimator=model,
            param_grid=param_grids[model_name],
            cv=5,
            scoring='accuracy',
            n_jobs=-1
        )

        grid_search.fit(X_train, y_train)
        best_model = grid_search.best_estimator_
        best_params = grid_search.best_params_

        valid_pred = best_model.predict(X_valid)
        valid_acc = accuracy_score(y_valid, valid_pred)

        print(f"Best Parameters for {model_name}:")
        for param, value in best_params.items():
            print(f"{param}: {value}")
        print(f"Best Validation Accuracy: {valid_acc:.4f}")

        tuned_models[model_name] = {
            'model': best_model,
            'params': best_params,
            'accuracy': valid_acc
        }
```

```
    return tuned_models

models = {
    'Random Forest': RandomForestClassifier(random_state=23),
    'Logistic Regression': LogisticRegression(max_iter=1000, random_state=23),
    'XGBoost': XGBClassifier(
        random_state=23,
        use_label_encoder=False,
        eval_metric='logloss'
    )
}

param_grids = {
    'Random Forest': {
        'n_estimators': [50, 60, 80, 100],
        'max_depth': [2, 5, 10, None]
    },
    'Logistic Regression': {
        'C': [0.001, 0.01, 0.1, 1, 10, 100],
        'penalty': ['l1', 'l2'],
        'solver': ['liblinear', 'saga']
    },
    'XGBoost': {
        'n_estimators': [50, 70, 80, 100],
        'max_depth': [3, 6, 9],
        'learning_rate': [0.001, 0.0001, 0.01, 0.1]
    }
}

for shark_idx in range(y_train.shape[1]):
    print(f"\n--- Hyperparameter Tuning for Shark {shark_idx + 1} ---")
    X_train_balanced, y_train_balanced = apply_smote(X_train, y_train[:, shark_idx])
    tuned_models = tune_hyperparameters(
        X_train_balanced, X_valid,
        y_train_balanced, y_valid[:, shark_idx],
        models, param_grids
    )
    best_model_name = max(tuned_models, key=lambda x: tuned_models[x]['accuracy'])
    print(f"\nBest Model for Shark {shark_idx + 1}: {best_model_name}")
```

```
--- Hyperparameter Tuning for Shark 1 ---
Applied SMOTE balancing

Tuning Random Forest
Best Parameters for Random Forest:
max_depth: None
n_estimators: 100
Best Validation Accuracy: 0.8439

Tuning Logistic Regression
Best Parameters for Logistic Regression:
C: 0.1
penalty: l1
solver: liblinear
Best Validation Accuracy: 0.6000

Tuning XGBoost
Best Parameters for XGBoost:
learning_rate: 0.1
max_depth: 9
n_estimators: 70
Best Validation Accuracy: 0.9073

Best Model for Shark 1: XGBoost

--- Hyperparameter Tuning for Shark 2 ---
Applied SMOTE balancing

Tuning Random Forest
Best Parameters for Random Forest:
max_depth: None
n_estimators: 100
Best Validation Accuracy: 0.7171

Tuning Logistic Regression
Best Parameters for Logistic Regression:
C: 0.01
penalty: l2
solver: saga
```

```
Best Validation Accuracy: 0.4878

Tuning XGBoost
Best Parameters for XGBoost:
learning_rate: 0.1
max_depth: 6
n_estimators: 80
Best Validation Accuracy: 0.7317

Best Model for Shark 2: XGBoost

--- Hyperparameter Tuning for Shark 3 ---
Applied SMOTE balancing

Tuning Random Forest
Best Parameters for Random Forest:
max_depth: None
n_estimators: 50
Best Validation Accuracy: 0.7220

Tuning Logistic Regression
Best Parameters for Logistic Regression:
C: 0.1
penalty: l2
solver: liblinear
Best Validation Accuracy: 0.4000

Tuning XGBoost
Best Parameters for XGBoost:
learning_rate: 0.1
max_depth: 9
n_estimators: 100
Best Validation Accuracy: 0.7659

Best Model for Shark 3: XGBoost

--- Hyperparameter Tuning for Shark 4 ---
Applied SMOTE balancing

Tuning Random Forest
Best Parameters for Random Forest:
max_depth: None
n_estimators: 50
Best Validation Accuracy: 0.8439

Tuning Logistic Regression
Best Parameters for Logistic Regression:
C: 100
penalty: l1
solver: liblinear
Best Validation Accuracy: 0.6488

Tuning XGBoost
Best Parameters for XGBoost:
learning_rate: 0.1
max_depth: 9
n_estimators: 100
Best Validation Accuracy: 0.8683

Best Model for Shark 4: XGBoost

--- Hyperparameter Tuning for Shark 5 ---
Applied SMOTE balancing

Tuning Random Forest
Best Parameters for Random Forest:
max_depth: None
n_estimators: 60
Best Validation Accuracy: 0.8683

Tuning Logistic Regression
Best Parameters for Logistic Regression:
C: 0.01
penalty: l2
solver: liblinear
Best Validation Accuracy: 0.6585

Tuning XGBoost
Best Parameters for XGBoost:
```

```
learning_rate: 0.1
max_depth: 9
n_estimators: 100
Best Validation Accuracy: 0.8390

Best Model for Shark 5: Random Forest

--- Hyperparameter Tuning for Shark 6 ---
Applied SMOTE balancing

Tuning Random Forest
Best Parameters for Random Forest:
max_depth: None
n_estimators: 100
Best Validation Accuracy: 0.8000

Tuning Logistic Regression
Best Parameters for Logistic Regression:
C: 100
penalty: l2
solver: liblinear
Best Validation Accuracy: 0.5659

Tuning XGBoost
Best Parameters for XGBoost:
learning_rate: 0.1
max_depth: 9
n_estimators: 100
Best Validation Accuracy: 0.8341

Best Model for Shark 6: XGBoost
```

In [ ]:
```python
models = {
    'Random Forest': RandomForestClassifier(n_estimators=100, max_depth=None, random_state=23),
    'Logistic Regression': LogisticRegression(
        C=0.1,
        penalty='l2',
        solver='liblinear',
        max_iter=1000,
        random_state=23
    ),
    'SVM': SVC(kernel='rbf', random_state=23),
    'XGBoost': XGBClassifier(
        n_estimators=100,
        max_depth=9,
        learning_rate=0.1,
        random_state=23,
        use_label_encoder=False,
        eval_metric='logloss'
    )
}
```

In [ ]:
```python
def analyze_results(results, shark_idx):
    best_model_name = max(
        results.items(),
        key=lambda x: x[1]['test_acc']
    )[0]
    print(f"\nBest Model for Shark {shark_idx + 1}: {best_model_name}")
    print(f"Test Accuracy: {results[best_model_name]['test_acc']:.4f}")

    return best_model_name
```

In [ ]:
```python
def print_detailed_metrics(X_train, X_test, y_train, y_test, shark_idx,
                           best_model_name, models):
    y_shark = y_train[:, shark_idx]
    X_train_balanced, y_train_balanced = apply_smote(X_train, y_shark)
    models[best_model_name].fit(X_train_balanced, y_train_balanced)
    y_pred = models[best_model_name].predict(X_test)
    print("\nConfusion Matrix:")
    cm = confusion_matrix(y_test[:, shark_idx], y_pred)
    print(cm)
    print("\nClassification Report:")
    print(classification_report(y_test[:, shark_idx], y_pred))
```

In [ ]:
```python
import os
import pickle
if not os.path.exists('saved_models_cls'):
    os.makedirs('saved_models_cls')
with open('saved_models_cls/scaler.pkl', 'wb') as f:
    pickle.dump(scaler, f)
```

In [ ]:
```python
def train_and_evaluate_shark(X_train, X_valid, X_test, y_train, y_valid, y_test,
                             shark_idx, models):
    results = {model_name: {'valid_acc': [], 'test_acc': []}
               for model_name in models.keys()}

    y_shark = y_train[:, shark_idx]

    unique, counts = np.unique(y_shark, return_counts=True)
    print(f"Original distribution: {dict(zip(unique, counts))}")

    X_train_balanced, y_train_balanced = apply_smote(X_train, y_shark)

    unique, counts = np.unique(y_train_balanced, return_counts=True)
    print(f"Final distribution: {dict(zip(unique, counts))}")

    for model_name, model in models.items():
        print(f"\nTraining {model_name} for Shark {shark_idx + 1}")

        model.fit(X_train_balanced, y_train_balanced)

        valid_pred = model.predict(X_valid)
        test_pred = model.predict(X_test)

        valid_acc = accuracy_score(y_valid[:, shark_idx], valid_pred)
        test_acc = accuracy_score(y_test[:, shark_idx], test_pred)
        results[model_name]['valid_acc'] = valid_acc
        results[model_name]['test_acc'] = test_acc

        print(f"{model_name} - Validation Accuracy: {valid_acc:.4f}, "
              f"Test Accuracy: {test_acc:.4f}")

    return results
```

In [ ]:
```python
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import roc_curve, auc, confusion_matrix
import numpy as np

def plot_all_results(results, shark_idx, X_test, y_test, models, best_model_name):
    fig = plt.figure(figsize=(20, 6))
    gs = fig.add_gridspec(1, 3)
    ax1 = fig.add_subplot(gs[0, 0])
    models_list = list(results.keys())
    valid_acc = [results[model]['valid_acc'] for model in models_list]
    test_acc = [results[model]['test_acc'] for model in models_list]
    x = np.arange(len(models_list))
    width = 0.35

    ax1.bar(x - width/2, valid_acc, width, label='Validation Accuracy', color='skyblue')
    ax1.bar(x + width/2, test_acc, width, label='Test Accuracy', color='lightgreen')
    ax1.set_ylabel('Accuracy')
    ax1.set_title(f'Model Performance Comparison\nShark {shark_idx + 1}')
    ax1.set_xticks(x)
    ax1.set_xticklabels(models_list, rotation=45)
    ax1.legend()
    ax2 = fig.add_subplot(gs[0, 1])
    best_model = models[best_model_name]
    y_pred = best_model.predict(X_test)
    cm = confusion_matrix(y_test[:, shark_idx], y_pred)

    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=['No Investment', 'Investment'],
                yticklabels=['No Investment', 'Investment'],
                ax=ax2)
    ax2.set_title(f'Confusion Matrix - {best_model_name}\nShark {shark_idx + 1}')
    ax2.set_ylabel('True Label')
    ax2.set_xlabel('Predicted Label')
    ax3 = fig.add_subplot(gs[0, 2])

    for model_name, model in models.items():
        if hasattr(model, 'predict_proba'):
            y_score = model.predict_proba(X_test)[:, 1]
        else:
            y_score = model.decision_function(X_test)

        fpr, tpr, _ = roc_curve(y_test[:, shark_idx], y_score)
        roc_auc = auc(fpr, tpr)

        ax3.plot(fpr, tpr, label=f'{model_name} (AUC = {roc_auc:.2f})')

    ax3.plot([0, 1], [0, 1], 'k--')
    ax3.set_xlim([0.0, 1.0])
    ax3.set_ylim([0.0, 1.05])
```

```
    ax3.set_xlabel('False Positive Rate')
    ax3.set_ylabel('True Positive Rate')
    ax3.set_title(f'ROC Curves\nShark {shark_idx + 1}')
    ax3.legend(loc="lower right")

    plt.tight_layout()
    return fig
```

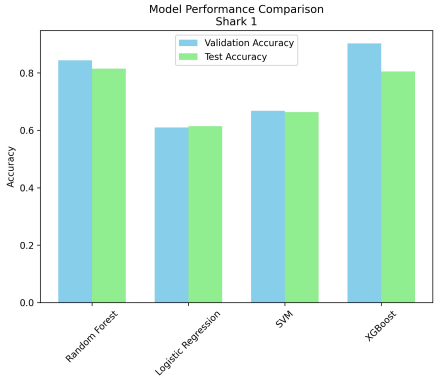
In [ ]:
```
all_results = []
for shark_idx in range(y_train.shape[1]):
    print(f"\nProcessing Shark {shark_idx + 1}")
    results = train_and_evaluate_shark(
        X_train, X_valid, X_test, y_train, y_valid, y_test, shark_idx, models
    )
    best_model_name = analyze_results(results, shark_idx)
    print_detailed_metrics(
        X_train, X_test, y_train, y_test, shark_idx, best_model_name, models
    )
    fig = plot_all_results(results, shark_idx, X_test, y_test, models, best_model_name)
    plt.savefig(f'shark_{shark_idx+1}_results.png', dpi=300, bbox_inches='tight')
    plt.close()
    best_model = models[best_model_name]
    with open(f'saved_models_cls/shark_{shark_idx+1}_model.pkl', 'wb') as f:
        pickle.dump(best_model, f)

    all_results.append({
        'shark_idx': shark_idx,
        'best_model': best_model_name,
        'results': results
    })
plt.figure(figsize=(12, 6))
shark_names = [f"Shark {i+1}" for i in range(len(all_results))]
best_accuracies = [result['results'][result['best_model']]['test_acc']
                   for result in all_results]
best_models = [result['best_model'] for result in all_results]
```

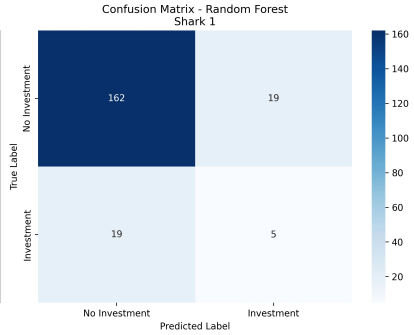
```
Processing Shark 1
Original distribution: {0: 861, 1: 94}
Applied SMOTE balancing
Final distribution: {0: 861, 1: 861}

Training Random Forest for Shark 1
Random Forest - Validation Accuracy: 0.8049, Test Accuracy: 0.7902

Training Logistic Regression for Shark 1
Logistic Regression - Validation Accuracy: 0.6195, Test Accuracy: 0.6244

Training SVM for Shark 1
SVM - Validation Accuracy: 0.6683, Test Accuracy: 0.6634

Training XGBoost for Shark 1
XGBoost - Validation Accuracy: 0.7463, Test Accuracy: 0.7512

Best Model for Shark 1: Random Forest
Test Accuracy: 0.7902
Applied SMOTE balancing

Confusion Matrix:
[[154  27]
 [ 16   8]]

Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.85      0.88       181
           1       0.23      0.33      0.27        24

    accuracy                           0.79       205
   macro avg       0.57      0.59      0.57       205
weighted avg       0.83      0.79      0.81       205


Processing Shark 2
Original distribution: {0: 779, 1: 176}
Applied SMOTE balancing
Final distribution: {0: 779, 1: 779}

Training Random Forest for Shark 2
```

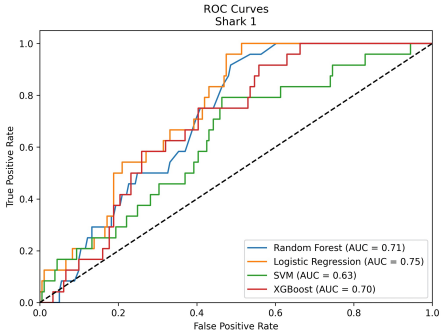
```
Random Forest - Validation Accuracy: 0.6732, Test Accuracy: 0.6683

Training Logistic Regression for Shark 2
Logistic Regression - Validation Accuracy: 0.4976, Test Accuracy: 0.5122

Training SVM for Shark 2
SVM - Validation Accuracy: 0.5415, Test Accuracy: 0.5268

Training XGBoost for Shark 2
XGBoost - Validation Accuracy: 0.6049, Test Accuracy: 0.6244

Best Model for Shark 2: Random Forest
Test Accuracy: 0.6683
Applied SMOTE balancing

Confusion Matrix:
[[128  38]
 [ 30   9]]

Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.77      0.79       166
           1       0.19      0.23      0.21        39

    accuracy                           0.67       205
   macro avg       0.50      0.50      0.50       205
weighted avg       0.69      0.67      0.68       205


Processing Shark 3
Original distribution: {0: 813, 1: 142}
Applied SMOTE balancing
Final distribution: {0: 813, 1: 813}

Training Random Forest for Shark 3
Random Forest - Validation Accuracy: 0.6927, Test Accuracy: 0.7171

Training Logistic Regression for Shark 3
Logistic Regression - Validation Accuracy: 0.4098, Test Accuracy: 0.4780

Training SVM for Shark 3
SVM - Validation Accuracy: 0.5268, Test Accuracy: 0.5707

Training XGBoost for Shark 3
XGBoost - Validation Accuracy: 0.7268, Test Accuracy: 0.6585

Best Model for Shark 3: Random Forest
Test Accuracy: 0.7171
Applied SMOTE balancing

Confusion Matrix:
[[138  28]
 [ 30   9]]

Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.83      0.83       166
           1       0.24      0.23      0.24        39

    accuracy                           0.72       205
   macro avg       0.53      0.53      0.53       205
weighted avg       0.71      0.72      0.71       205


Processing Shark 4
Original distribution: {0: 868, 1: 87}
Applied SMOTE balancing
Final distribution: {0: 868, 1: 868}

Training Random Forest for Shark 4
Random Forest - Validation Accuracy: 0.8098, Test Accuracy: 0.7415

Training Logistic Regression for Shark 4
Logistic Regression - Validation Accuracy: 0.6488, Test Accuracy: 0.5951

Training SVM for Shark 4
SVM - Validation Accuracy: 0.6390, Test Accuracy: 0.6683
```

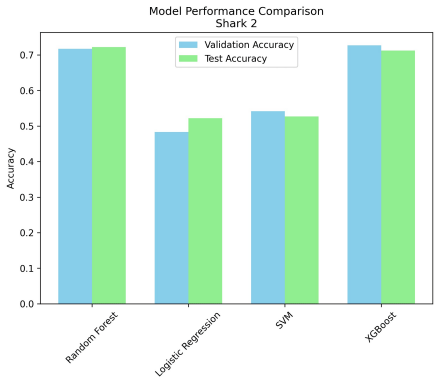
```
Training XGBoost for Shark 4
XGBoost - Validation Accuracy: 0.7951, Test Accuracy: 0.7366

Best Model for Shark 4: Random Forest
Test Accuracy: 0.7415
Applied SMOTE balancing

Confusion Matrix:
[[148  37]
 [ 16   4]]

Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.80      0.85       185
           1       0.10      0.20      0.13        20

    accuracy                           0.74       205
   macro avg       0.50      0.50      0.49       205
weighted avg       0.82      0.74      0.78       205


Processing Shark 5
Original distribution: {0: 867, 1: 88}
Applied SMOTE balancing
Final distribution: {0: 867, 1: 867}

Training Random Forest for Shark 5
Random Forest - Validation Accuracy: 0.8244, Test Accuracy: 0.7951

Training Logistic Regression for Shark 5
Logistic Regression - Validation Accuracy: 0.6341, Test Accuracy: 0.6390

Training SVM for Shark 5
SVM - Validation Accuracy: 0.6341, Test Accuracy: 0.6878

Training XGBoost for Shark 5
XGBoost - Validation Accuracy: 0.7463, Test Accuracy: 0.7220

Best Model for Shark 5: Random Forest
Test Accuracy: 0.7951
Applied SMOTE balancing

Confusion Matrix:
[[156  36]
 [  6   7]]

Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.81      0.88       192
           1       0.16      0.54      0.25        13

    accuracy                           0.80       205
   macro avg       0.56      0.68      0.57       205
weighted avg       0.91      0.80      0.84       205


Processing Shark 6
Original distribution: {0: 869, 1: 86}
Applied SMOTE balancing
Final distribution: {0: 869, 1: 869}

Training Random Forest for Shark 6
Random Forest - Validation Accuracy: 0.7512, Test Accuracy: 0.7902

Training Logistic Regression for Shark 6
Logistic Regression - Validation Accuracy: 0.5659, Test Accuracy: 0.5756

Training SVM for Shark 6
SVM - Validation Accuracy: 0.6390, Test Accuracy: 0.6341

Training XGBoost for Shark 6
XGBoost - Validation Accuracy: 0.7171, Test Accuracy: 0.7415

Best Model for Shark 6: Random Forest
Test Accuracy: 0.7902
Applied SMOTE balancing
```

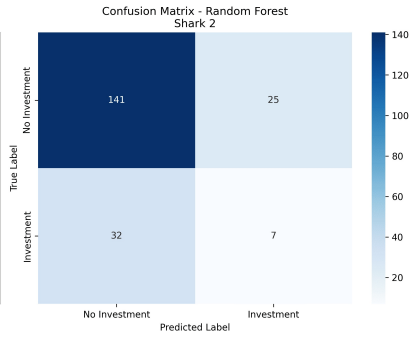
```
Confusion Matrix:
[[161  28]
 [ 15   1]]

Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.85      0.88       189
           1       0.03      0.06      0.04        16

    accuracy                           0.79       205
   macro avg       0.47      0.46      0.46       205
weighted avg       0.85      0.79      0.82       205

<Figure size 1200x600 with 0 Axes>
```

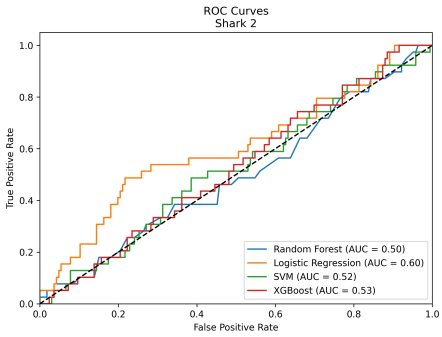


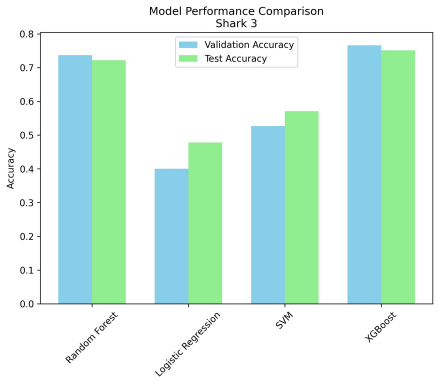



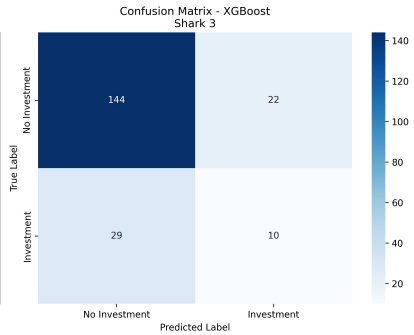



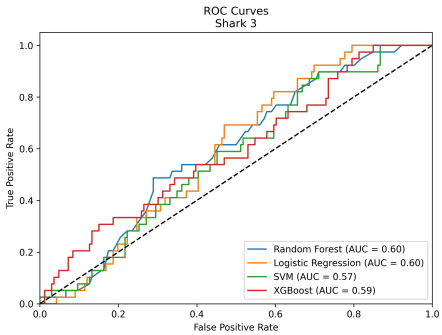



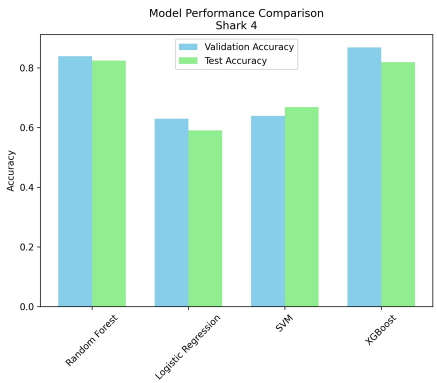



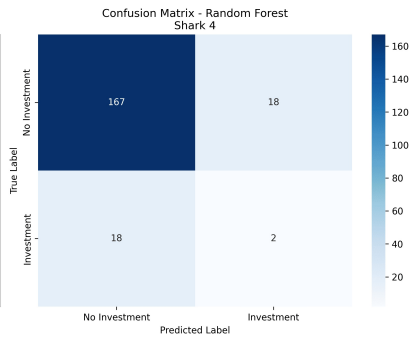



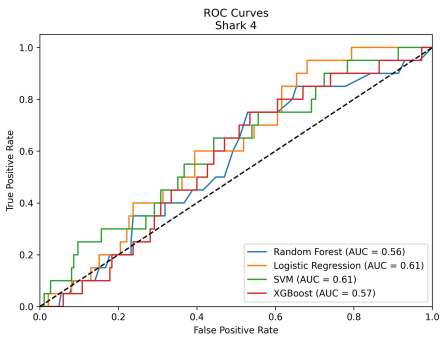



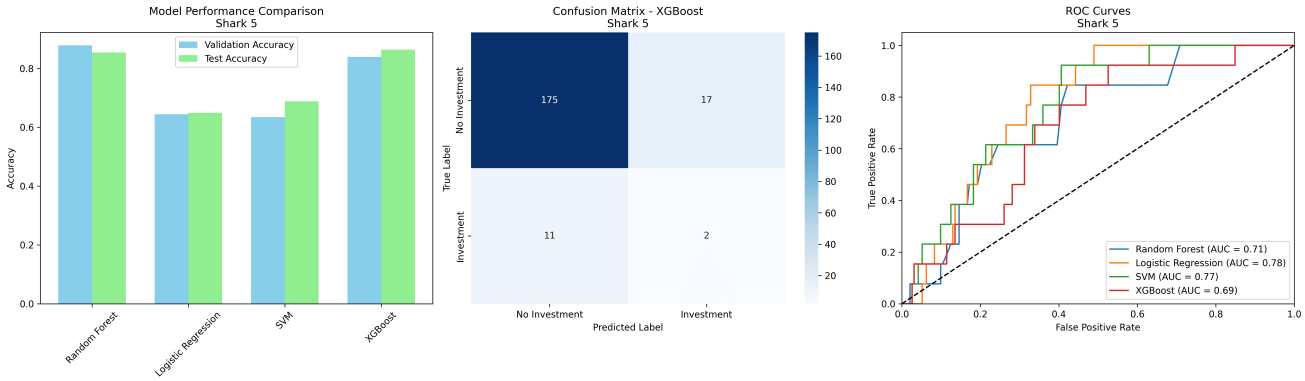



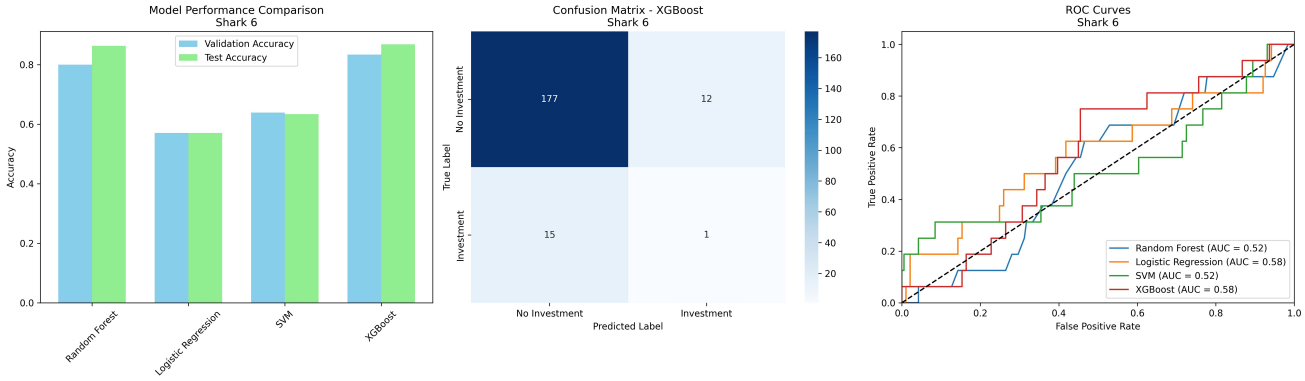






In [ ]:
```python
def load_models_and_predict(new_data):
    with open('saved_models_cls/scaler.pkl', 'rb') as f:
        saved_scaler = pickle.load(f)
    with open('saved_models_cls/metadata.pkl', 'rb') as f:
        saved_results = pickle.load(f)
    X_scaled = saved_scaler.transform(new_data)
    n_sharks = len(saved_results)
    predictions = np.zeros((len(new_data), n_sharks))
    for result in saved_results:
        shark_idx = result['shark_idx']
        with open(f'saved_models_cls/shark_{shark_idx+1}_model.pkl', 'rb') as f:
            model = pickle.load(f)
        predictions[:, shark_idx] = model.predict(X_scaled)

    return predictions
```

In [ ]:
```python
with open('saved_models_cls/metadata.pkl', 'wb') as f:
    pickle.dump(all_results, f)
```

Regression

In [ ]:
```python
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
from xgboost import XGBRegressor
from sklearn.svm import SVR
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
import warnings
import matplotlib.pyplot as plt
import seaborn as sns
warnings.filterwarnings('ignore')
y_reg = np.array([np.array(x) for x in df_regres['Sharks Investment Amounts']])
X_reg = df_regres.drop(columns=['Sharks Investment Amounts'])
```

In [ ]:
```python
print(type(X_reg))
```

In [ ]:
```python
df_regres.head()
```

Now we use the dataframe for regression created earlier for the regression task by transforming the features and target variables. A copy of the dataset df_regres_clean is created, and the Sharks Invested column is expanded into six separate columns, each indicating whether a specific shark invested binary values. These new columns are appended to the feature set X_reg, and other irrelevant columns, such as Sharks Invested and Sharks Investment Amounts, are dropped. The features are then standardized using StandardScaler to ensure consistent scaling for numerical stability during model training. The target variable, y_reg, is extracted as a NumPy array, representing the actual investment amounts for each shark in a structured format suitable for regression.

In [ ]:
```
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler


df_regres_clean = df_regres.copy()
sharks_invested = pd.DataFrame(df_regres_clean['Sharks Invested'].tolist(),
                               columns=[f'Shark_{i+1}_Invested' for i in range(6)])
X_reg = df_regres_clean.drop(columns=['Sharks Invested', 'Sharks Investment Amounts'])
X_reg = pd.concat([X_reg, sharks_invested], axis=1)

scaler = StandardScaler()
X_reg = scaler.fit_transform(X_reg)
y_reg = np.array([np.array(x) for x in df_regres['Sharks Investment Amounts']])
```

In [ ]:
```
X_train, X_temp, y_train, y_temp = train_test_split(X_reg, y_reg, test_size=0.3, random_state=42)
X_valid, X_test, y_valid, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)
```

We chose Random Forest and XGBoost for regression because they are powerful, robust, and widely used ensemble learning methods that perform exceptionally well on complex, non-linear datasets. We have already seen them performing amazingly in classification tasks and hence we are gonna use them for regression task

In [ ]:
```
models = {
    'Random Forest': RandomForestRegressor(n_estimators=100, max_depth=None, random_state=23),
    'XGBoost': XGBRegressor(
        n_estimators=100,
        max_depth=9,
        learning_rate=0.1,
        random_state=23
    )
}
```

For training regression models we are using metrics like MSE or Mean Squared Error to measure the average squared prediction error, RMSE or Root Mean Squared Error for interpreting errors in the same scale as the target variable, MAE or Mean Absolute Error for assessing average absolute errors, and $R^2$ or Coefficient of Determination to evaluate how well the model explains variance in the target.

Also the **train_and_evaluate_shark_regression** function trains models (e.g., Random Forest and XGBoost) for a specific shark, calculates predictions, and evaluates them using these metrics on validation and test sets. The analyze_regression_results function identifies the best-performing model for each shark by comparing test RMSE and summarizes its performance, ensuring the selection of the most effective model for accurate investment prediction.

The reason for choosing rmse it gives an interpretable measure of the predictive power of the regression model, also it penalizes larger errors because of the squaring part.

In [ ]:
```
def evaluate_regression_metrics(y_true, y_pred):
    mse = mean_squared_error(y_true, y_pred)
    rmse = np.sqrt(mse)
    mae = mean_absolute_error(y_true, y_pred)
    r2 = r2_score(y_true, y_pred)

    return {
        'MSE': mse,
        'RMSE': rmse,
        'MAE': mae,
        'R2': r2
    }

def train_and_evaluate_shark_regression(X_train, X_valid, X_test, y_train, y_valid, y_test, shark_idx, models):
    results = {model_name: {'valid_metrics': {}, 'test_metrics': {}}
               for model_name in models.keys()}

    y_shark_train = y_train[:, shark_idx]
    y_shark_valid = y_valid[:, shark_idx]
    y_shark_test = y_test[:, shark_idx]

    print(f"\nTraining models for Shark {shark_idx + 1}")
    print(f"Average investment amount: ${np.mean(y_shark_train):.2f}")
    print(f"Max investment amount: ${np.max(y_shark_train):.2f}")

    for model_name, model in models.items():
        print(f"\nTraining {model_name}")
        model.fit(X_train, y_shark_train)
        valid_pred = model.predict(X_valid)
        test_pred = model.predict(X_test)
        valid_metrics = evaluate_regression_metrics(y_shark_valid, valid_pred)
        test_metrics = evaluate_regression_metrics(y_shark_test, test_pred)

        results[model_name]['valid_metrics'] = valid_metrics
        results[model_name]['test_metrics'] = test_metrics
```

```
        print(f"{model_name} Results:")
        print(f"Validation RMSE: ${valid_metrics['RMSE']:.2f}")
        print(f"Test RMSE: ${test_metrics['RMSE']:.2f}")
        print(f"Test R2 Score: {test_metrics['R2']:.4f}")

    return results

def analyze_regression_results(results, shark_idx):
    best_model_name = min(
        results.items(),
        key=lambda x: x[1]['test_metrics']['RMSE']
    )[0]

    print(f"\nBest Model for Shark {shark_idx + 1}: {best_model_name}")
    print(f"Test RMSE: ${results[best_model_name]['test_metrics']['RMSE']:.2f}")
    print(f"Test R2: {results[best_model_name]['test_metrics']['R2']:.4f}")

    return best_model_name
```

In [ ]:

```
def print_detailed_regression_metrics(X_train, X_test, y_train, y_test, shark_idx,
                                      best_model_name, models):
    y_shark_train = y_train[:, shark_idx]
    y_shark_test = y_test[:, shark_idx]
    model = models[best_model_name]
    model.fit(X_train, y_shark_train)
    y_pred = model.predict(X_test)
    metrics = evaluate_regression_metrics(y_shark_test, y_pred)

    print("\nDetailed Metrics:")
    print(f"Mean Squared Error: ${metrics['MSE']:.2f}")
    print(f"Root Mean Squared Error: ${metrics['RMSE']:.2f}")
    print(f"Mean Absolute Error: ${metrics['MAE']:.2f}")
    print(f"R² Score: {metrics['R2']:.4f}")
```

In [ ]:

```
def plot_regression_results(results, shark_idx, X_test, y_test, models, best_model_name):
    """Create a comprehensive visualization of regression model performance"""
    fig = plt.figure(figsize=(20, 10))
    gs = fig.add_gridspec(2, 3)
    ax1 = fig.add_subplot(gs[0, 0])
    models_list = list(results.keys())
    valid_rmse = [results[model]['valid_metrics']['RMSE'] for model in models_list]
    test_rmse = [results[model]['test_metrics']['RMSE'] for model in models_list]
    x = np.arange(len(models_list))
    width = 0.35

    ax1.bar(x - width/2, valid_rmse, width, label='Validation RMSE', color='skyblue')
    ax1.bar(x + width/2, test_rmse, width, label='Test RMSE', color='lightgreen')
    ax1.set_ylabel('RMSE ($)')
    ax1.set_title(f'RMSE Comparison\nShark {shark_idx + 1}')
    ax1.set_xticks(x)
    ax1.set_xticklabels(models_list, rotation=45)
    ax1.legend()

    ax2 = fig.add_subplot(gs[0, 1])
    valid_r2 = [results[model]['valid_metrics']['R2'] for model in models_list]
    test_r2 = [results[model]['test_metrics']['R2'] for model in models_list]
    ax2.bar(x - width/2, valid_r2, width, label='Validation R²', color='skyblue')
    ax2.bar(x + width/2, test_r2, width, label='Test R²', color='lightgreen')
    ax2.set_ylabel('R² Score')
    ax2.set_title(f'R² Score Comparison\nShark {shark_idx + 1}')
    ax2.set_xticks(x)
    ax2.set_xticklabels(models_list, rotation=45)
    ax2.legend()

    ax3 = fig.add_subplot(gs[0, 2])
    best_model = models[best_model_name]
    y_pred = best_model.predict(X_test)
    ax3.scatter(y_test[:, shark_idx], y_pred, alpha=0.5)
    max_val = max(np.max(y_test[:, shark_idx]), np.max(y_pred))
    ax3.plot([0, max_val], [0, max_val], 'r--')
    ax3.set_xlabel('Actual Investment ($)')
    ax3.set_ylabel('Predicted Investment ($)')
    ax3.set_title(f'Predicted vs Actual - {best_model_name}\nShark {shark_idx + 1}')

    ax4 = fig.add_subplot(gs[1, 0])
    residuals = y_test[:, shark_idx] - y_pred
    ax4.scatter(y_pred, residuals, alpha=0.5)
    ax4.axhline(y=0, color='r', linestyle='--')
    ax4.set_xlabel('Predicted Investment ($)')
    ax4.set_ylabel('Residuals ($)')
    ax4.set_title(f'Residual Plot - {best_model_name}\nShark {shark_idx + 1}')
```

```
    ax5 = fig.add_subplot(gs[1, 1])
    sns.histplot(residuals, kde=True, ax=ax5)
    ax5.set_xlabel('Residuals ($)')
    ax5.set_ylabel('Count')
    ax5.set_title(f'Error Distribution - {best_model_name}\nShark {shark_idx + 1}')

    plt.tight_layout()
    return fig
```

In [ ]:
```
import os
import pickle

if not os.path.exists('saved_regression_models'):
    os.makedirs('saved_regression_models')

with open('saved_regression_models/scaler.pkl', 'wb') as f:
    pickle.dump(scaler, f)
```

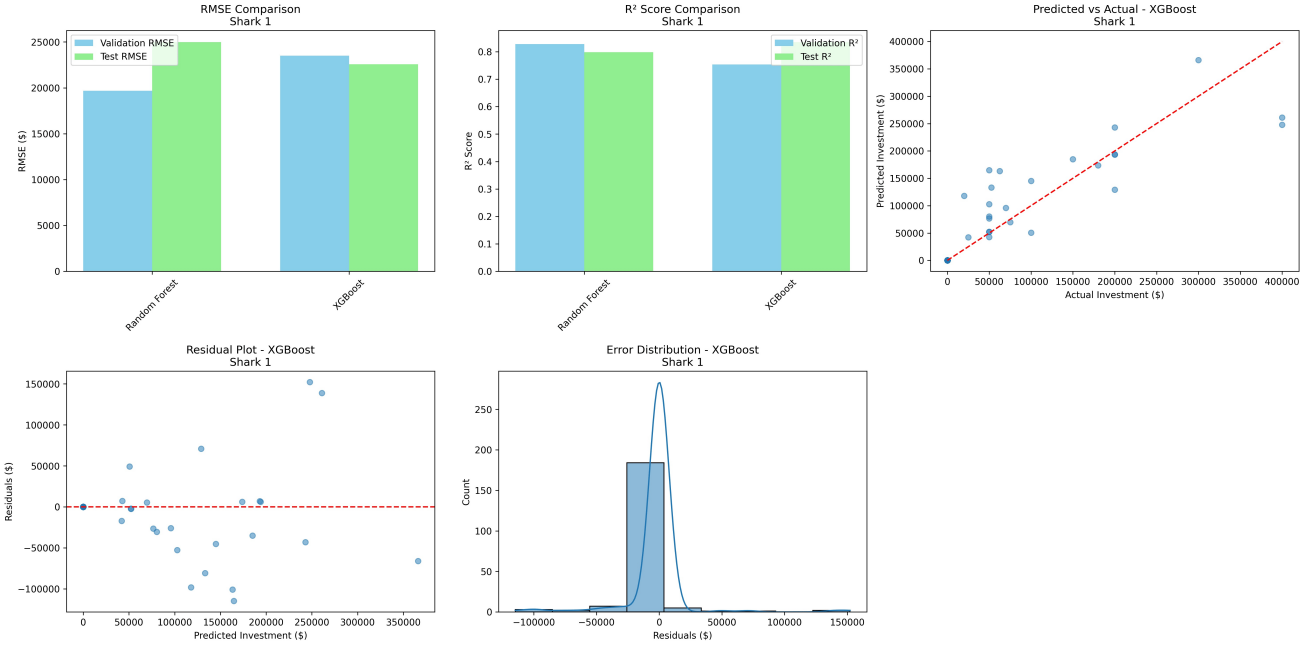
In [ ]:
```
all_regression_results = []
for shark_idx in range(y_train.shape[1]):
    print(f"\nProcessing Shark {shark_idx + 1}")
    results = train_and_evaluate_shark_regression(
        X_train, X_valid, X_test, y_train, y_valid, y_test, shark_idx, models
    )
    best_model_name = analyze_regression_results(results, shark_idx)
    print_detailed_regression_metrics(
        X_train, X_test, y_train, y_test, shark_idx, best_model_name, models
    )
    fig = plot_regression_results(results, shark_idx, X_test, y_test, models, best_model_name)
    plt.savefig(f'shark_{shark_idx+1}_regression_results.png', dpi=300, bbox_inches='tight')
    plt.close()
    best_model = models[best_model_name]
    with open(f'saved_regression_models/shark_{shark_idx+1}_regression_model.pkl', 'wb') as f:
        pickle.dump(best_model, f)

    all_regression_results.append({
        'shark_idx': shark_idx,
        'best_model': best_model_name,
        'results': results
    })
```

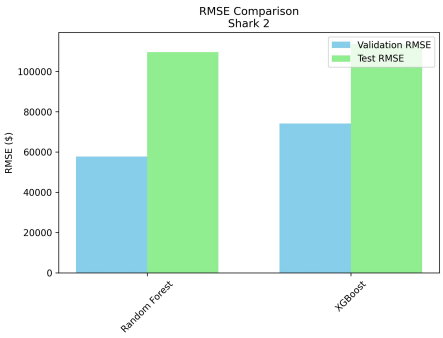
```
Processing Shark 1

Training models for Shark 1
Average investment amount: $15209.42
Max investment amount: $700000.00

Training Random Forest
Random Forest Results:
Validation RMSE: $19679.36
Test RMSE: $24994.13
Test R2 Score: 0.7981

Training XGBoost
XGBoost Results:
Validation RMSE: $23512.55
Test RMSE: $22570.87
Test R2 Score: 0.8353

Best Model for Shark 1: XGBoost
Test RMSE: $22570.87
Test R2: 0.8353

Detailed Metrics:
Mean Squared Error: $509444346.20
Root Mean Squared Error: $22570.87
Mean Absolute Error: $5774.53
R² Score: 0.8353

Processing Shark 2

Training models for Shark 2
Average investment amount: $48763.70
Max investment amount: $2000000.00

Training Random Forest
Random Forest Results:
Validation RMSE: $57750.41
```

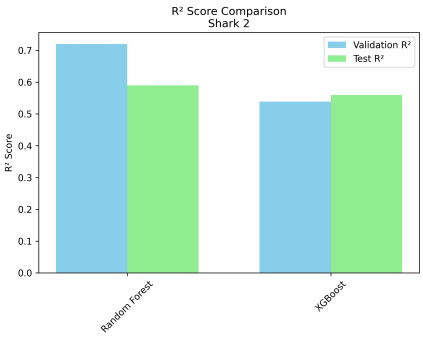
```
Test RMSE: $109626.95
Test R2 Score: 0.5898

Training XGBoost
XGBoost Results:
Validation RMSE: $74114.59
Test RMSE: $113666.52
Test R2 Score: 0.5590

Best Model for Shark 2: Random Forest
Test RMSE: $109626.95
Test R2: 0.5898

Detailed Metrics:
Mean Squared Error: $12018068726.26
Root Mean Squared Error: $109626.95
Mean Absolute Error: $24296.24
R² Score: 0.5898

Processing Shark 3

Training models for Shark 3
Average investment amount: $30222.51
Max investment amount: $1000000.00

Training Random Forest
Random Forest Results:
Validation RMSE: $34736.88
Test RMSE: $85036.95
Test R2 Score: 0.6453

Training XGBoost
XGBoost Results:
Validation RMSE: $44220.53
Test RMSE: $93039.88
Test R2 Score: 0.5754

Best Model for Shark 3: Random Forest
Test RMSE: $85036.95
Test R2: 0.6453

Detailed Metrics:
Mean Squared Error: $7231282369.25
Root Mean Squared Error: $85036.95
Mean Absolute Error: $20273.50
R² Score: 0.6453

Processing Shark 4

Training models for Shark 4
Average investment amount: $27136.82
Max investment amount: $5000000.00

Training Random Forest
Random Forest Results:
Validation RMSE: $48559.03
Test RMSE: $216530.50
Test R2 Score: -5.0121

Training XGBoost
XGBoost Results:
Validation RMSE: $43881.02
Test RMSE: $233194.26
Test R2 Score: -5.9731

Best Model for Shark 4: Random Forest
Test RMSE: $216530.50
Test R2: -5.0121

Detailed Metrics:
Mean Squared Error: $46885456820.26
Root Mean Squared Error: $216530.50
Mean Absolute Error: $22523.63
R² Score: -5.0121

Processing Shark 5

Training models for Shark 5
Average investment amount: $17177.84
```

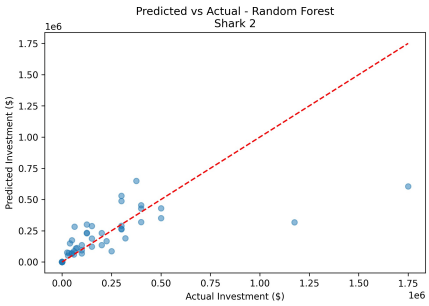
```
Max investment amount: $3000000.00

Training Random Forest
Random Forest Results:
Validation RMSE: $8751.28
Test RMSE: $21579.29
Test R2 Score: 0.7297

Training XGBoost
XGBoost Results:
Validation RMSE: $6063.93
Test RMSE: $8524.34
Test R2 Score: 0.9578

Best Model for Shark 5: XGBoost
Test RMSE: $8524.34
Test R2: 0.9578

Detailed Metrics:
Mean Squared Error: $72664337.44
Root Mean Squared Error: $8524.34
Mean Absolute Error: $1442.77
R² Score: 0.9578

Processing Shark 6

Training models for Shark 6
Average investment amount: $22449.39
Max investment amount: $2500000.00

Training Random Forest
Random Forest Results:
Validation RMSE: $32823.19
Test RMSE: $27258.33
Test R2 Score: 0.8910

Training XGBoost
XGBoost Results:
Validation RMSE: $39101.91
Test RMSE: $46840.22
Test R2 Score: 0.6781

Best Model for Shark 6: Random Forest
Test RMSE: $27258.33
Test R2: 0.8910

Detailed Metrics:
Mean Squared Error: $743016330.21
Root Mean Squared Error: $27258.33
Mean Absolute Error: $4412.15
R² Score: 0.8910
```

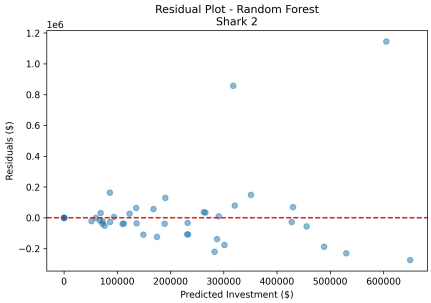
Comparison Between XGBoost regression and Random Forest Regression

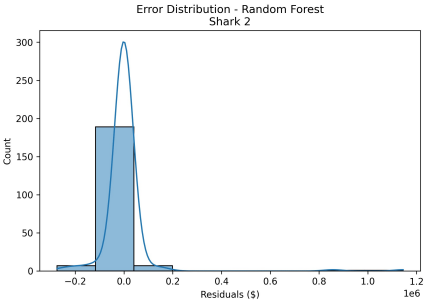


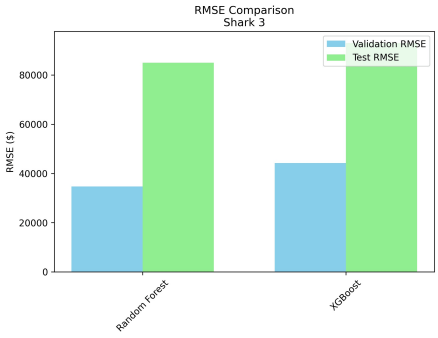

RMSE Comparison
Shark 2
Validation RMSE
Test RMSE
RMSE ($)
Random Forest
XGBoost

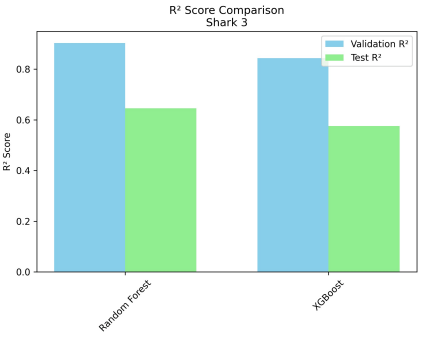

R² Score Comparison
Shark 2
Validation R²
Test R²
R² Score
Random Forest
XGBoost

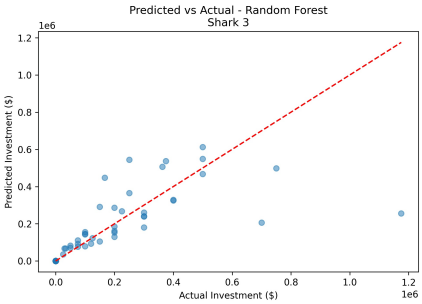

Predicted vs Actual - Random Forest
Shark 2
Predicted Investment ($)
Actual Investment ($)

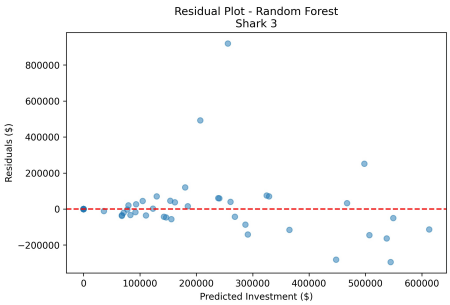

Residual Plot - Random Forest
Shark 2
Residuals ($)
Predicted Investment ($)

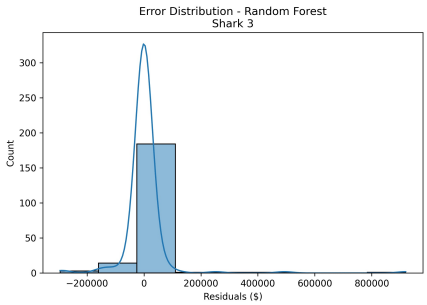

Error Distribution - Random Forest
Shark 2
Count
Residuals ($)

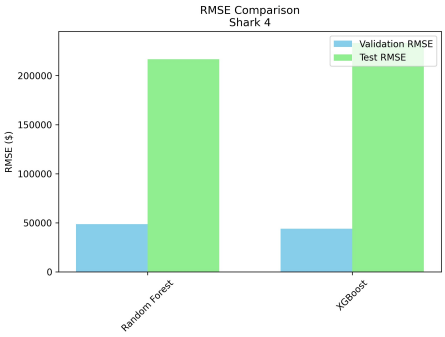

RMSE Comparison
Shark 3
Validation RMSE
Test RMSE
RMSE ($)
Random Forest
XGBoost

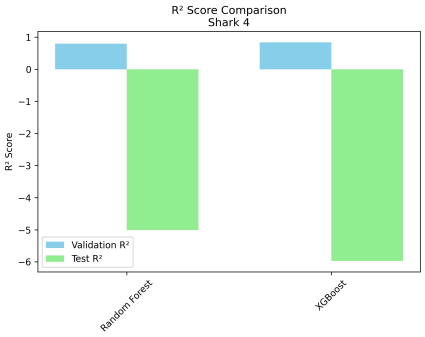

R² Score Comparison
Shark 3
Validation R²
Test R²
R² Score
Random Forest
XGBoost

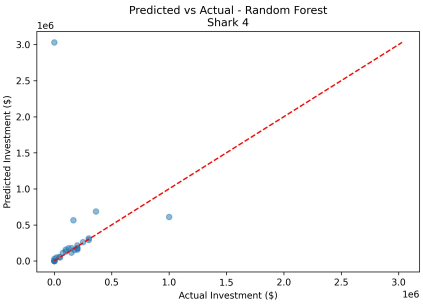

Predicted vs Actual - Random Forest
Shark 3
Predicted Investment ($)
Actual Investment ($)

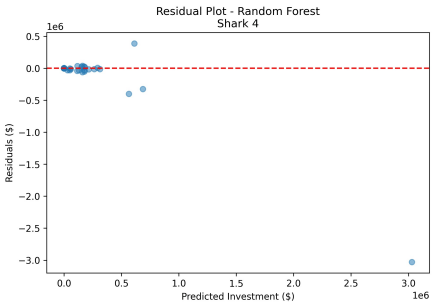

Residual Plot - Random Forest
Shark 3
Residuals ($)
Predicted Investment ($)

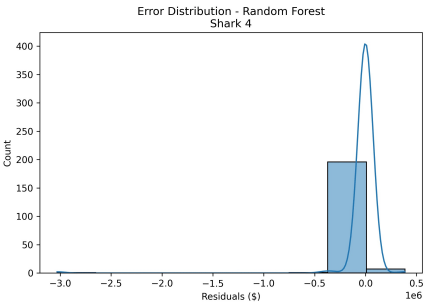

Error Distribution - Random Forest
Shark 3
Count
Residuals ($)

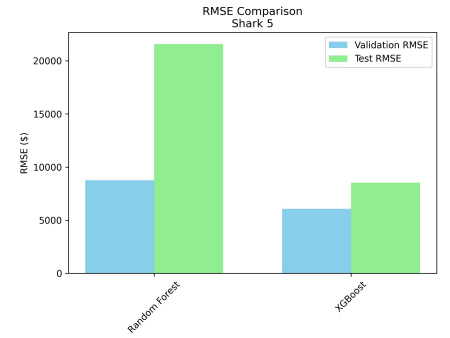

RMSE Comparison
Shark 4
Validation RMSE
Test RMSE
RMSE ($)
Random Forest
XGBoost

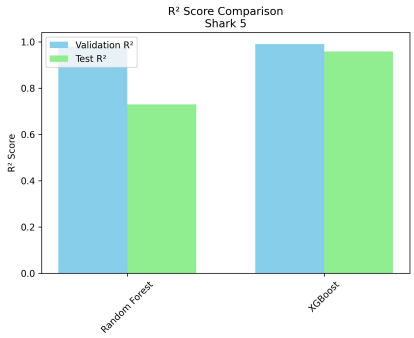

R² Score Comparison
Shark 4
Validation R²
Test R²
R² Score
Random Forest
XGBoost

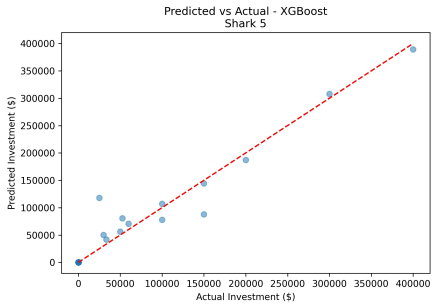

Predicted vs Actual - Random Forest
Shark 4
Predicted Investment ($)
Actual Investment ($)

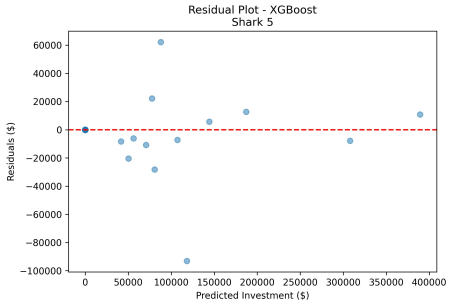

Residual Plot - Random Forest
Shark 4
Residuals ($)
Predicted Investment ($)

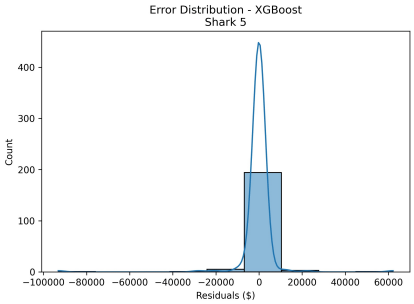

Error Distribution - Random Forest
Shark 4
Count
Residuals ($)

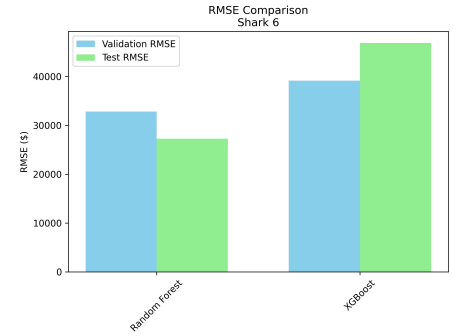


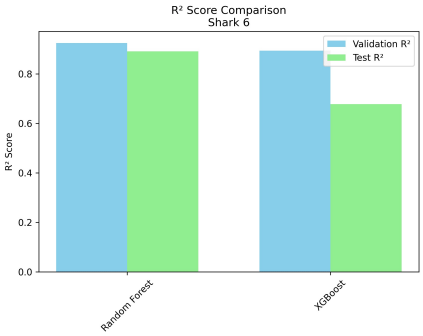



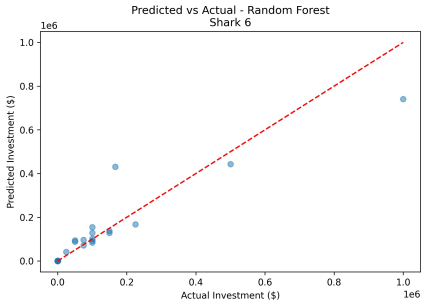



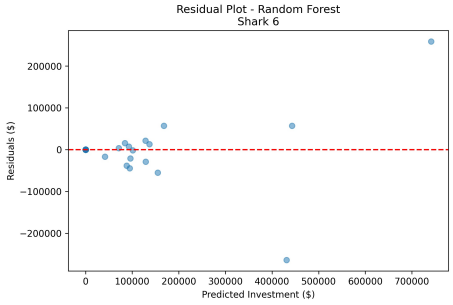



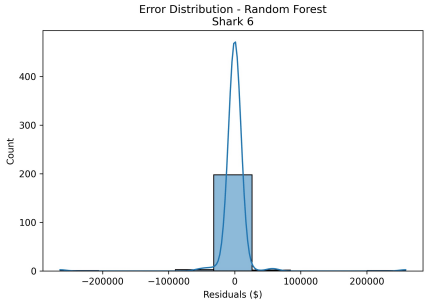




In [ ]:
```python
with open('saved_regression_models/metadata.pkl', 'wb') as f:
    pickle.dump(all_regression_results, f)

def load_regression_models_and_predict(new_data):
    with open('saved_regression_models/scaler.pkl', 'rb') as f:
        saved_scaler = pickle.load(f)
    with open('saved_regression_models/metadata.pkl', 'rb') as f:
        saved_results = pickle.load(f)
    X_scaled = saved_scaler.transform(new_data)
    n_sharks = len(saved_results)
    predictions = np.zeros((len(new_data), n_sharks))
    for result in saved_results:
        shark_idx = result['shark_idx']
        with open(f'saved_regression_models/shark_{shark_idx+1}_regression_model.pkl', 'rb') as f:
            model = pickle.load(f)
        predictions[:, shark_idx] = model.predict(X_scaled)
    return predictions
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: