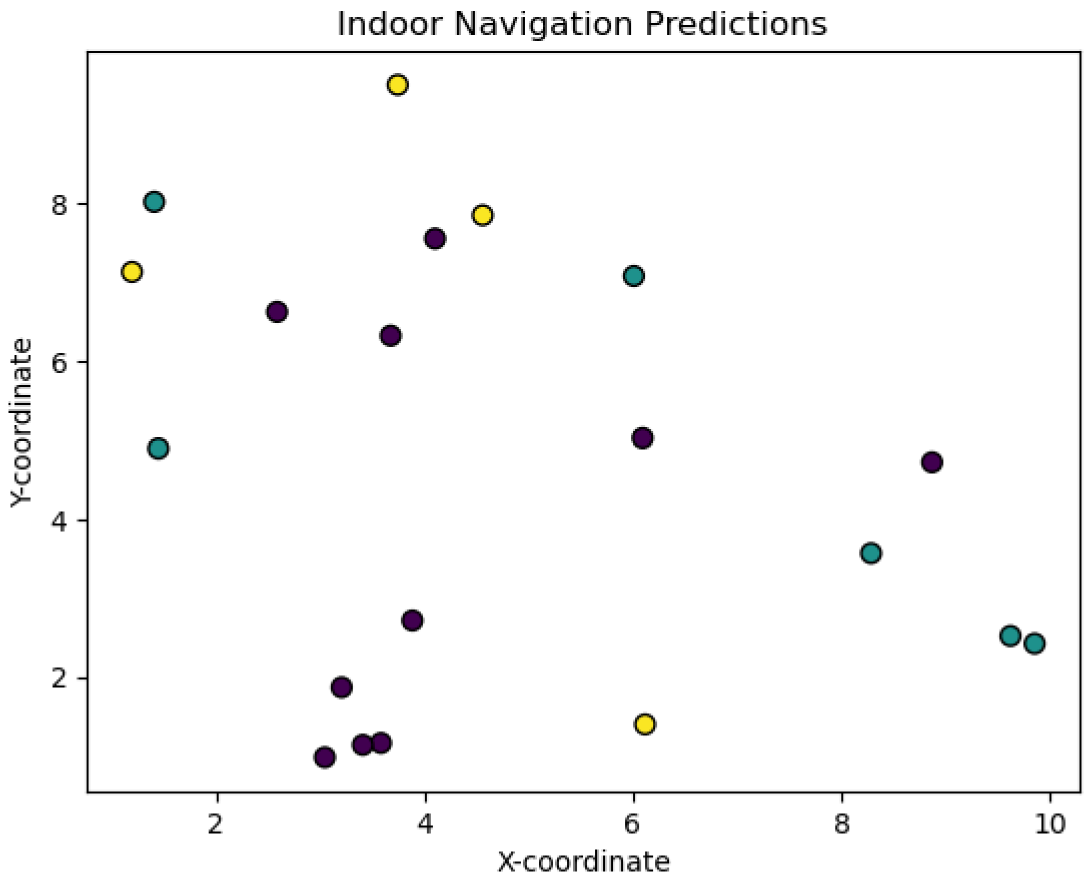
1. Maps And Navigation

In [1]:

```python
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

np.random.seed(42)
num_samples = 100
features = np.random.rand(num_samples, 2) * 10
labels = np.random.choice([0, 1, 2], size=num_samples)

X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size=0.2, random_state=42)

knn_classifier = KNeighborsClassifier(n_neighbors=3)
knn_classifier.fit(X_train, y_train)

y_pred = knn_classifier.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

plt.scatter(X_test[:, 0], X_test[:, 1], c=y_pred, cmap='viridis', marker='o', s=50, edgecolors='k')
plt.title('Indoor Navigation Predictions')
plt.xlabel('X-coordinate')
plt.ylabel('Y-coordinate')
plt.show()
```

```
Accuracy: 0.30
```



3. Write a AI Program Code for Implementing an Algorithm that Gives auto -suggestions in Word Processor.

In [3]:
```
class TrieNode:
    def __init__(self):
        self.children = {}
        self.is_end_of_word = False

class AutoSuggest:
    def __init__(self):
        self.root = TrieNode()

    def insert_word(self, word):
        node = self.root
        for char in word:
            if char not in node.children:
                node.children[char] = TrieNode()
            node = node.children[char]
        node.is_end_of_word = True

    def build_trie(self, words):
        for word in words:
            self.insert_word(word)

    def auto_suggest(self, prefix):
        node = self.root
        suggestions = []
        for char in prefix:
            if char not in node.children:
                return suggestions
            node = node.children[char]

        self._dfs(node, prefix, suggestions)
        return suggestions

    def _dfs(self, node, current_prefix, suggestions):
        if node.is_end_of_word:
            suggestions.append(current_prefix)
        for char, child_node in node.children.items():
            self._dfs(child_node, current_prefix + char, suggestions)

# Example usage with user input:
word_suggester = AutoSuggest()
words = ["apple", "app", "application", "banana", "bat", "batman"]
```

```python
word_suggester.build_trie(words)
while True:
    user_input = input("Enter a prefix (or 'exit' to quit): ")
    if user_input.lower() == 'exit':
        break
    suggestions = word_suggester.auto_suggest(user_input)
    print(f"Suggestions for '{user_input}': {suggestions}")
```

```
Enter a prefix (or 'exit' to quit): ap
Suggestions for 'ap': ['app', 'apple', 'application']
Enter a prefix (or 'exit' to quit): bat
Suggestions for 'bat': ['bat', 'batman']
Enter a prefix (or 'exit' to quit): exit
```

5A. Program For a Basic College Information Chatbot

In [4]:
```python
def process_input(user_input):
    # Preprocess user input (e.g., remove punctuation, convert to lowercase)
    return user_input.lower()

def retrieve_college_info(query):
    # Placeholder function to retrieve college information (e.g., from a database)
    # In a real implementation, this function would query a database or scrape information from websites
    # For demonstration purposes, return mock data
    mock_data = {
        "name": "Example University",
        "location": "Example City",
        "programs": ["Computer Science", "Engineering", "Business"],
        "admission_requirements": "GPA of 3.0 or higher, SAT score of 1200 or higher",
        "facilities": "State-of-the-art labs, library, sports facilities",
        "website": "www.exampleuniversity.com"
    }
    return mock_data

def generate_response(college_info):
    # Generate a response based on the retrieved college information
    response = f"Here is some information about {college_info['name']}: \n"
    response += f"Location: {college_info['location']}\n"
    response += f"Programs offered: {', '.join(college_info['programs'])}\n"
    response += f"Admission Requirements: {college_info['admission_requirements']}\n"
    response += f"Facilities: {college_info['facilities']}\n"
    response += f"For more information, visit {college_info['website']}"
    return response

# Main function to run the chatbot
def college_info_chatbot():
    print("Welcome to the College Information Chatbot!")
    print("Ask me anything about colleges or universities.")

    while True:
        user_input = input("You: ")
        processed_input = process_input(user_input)

        # Check for exit command
        if processed_input == "exit":
            print("Goodbye!")
            break

        # Retrieve college information based on user input
        college_info = retrieve_college_info(processed_input)

        if college_info:
            response = generate_response(college_info)
            print("Bot:", response)
        else:
            print("Bot: Sorry, I couldn't find information about that college.")

# Run the chatbot
college_info_chatbot()
```

```
Welcome to the College Information Chatbot!
Ask me anything about colleges or universities.
You: Tell me about Stanford University
Bot: Here is some information about Example University:
Location: Example City
Programs offered: Computer Science, Engineering, Business
Admission Requirements: GPA of 3.0 or higher, SAT score of 1200 or higher
Facilities: State-of-the-art labs, library, sports facilities
For more information, visit www.exampleuniversity.com
You: What programs does Harvard offer?
Bot: Here is some information about Example University:
Location: Example City
Programs offered: Computer Science, Engineering, Business
Admission Requirements: GPA of 3.0 or higher, SAT score of 1200 or higher
Facilities: State-of-the-art labs, library, sports facilities
For more information, visit www.exampleuniversity.com
You: exit
Goodbye!
```

5B. Program For a Software Installation Chatbot

In [5]:
```python
def process_input(user_input):
    return user_input.lower()

def install_software(software_name, operating_system):
    return f"Please follow these steps to install {software_name} on {operating_system}:\nStep 1: Download the ins

# Main function to run the chatbot
def software_installation_chatbot():
    print("Welcome to the Software Installation Chatbot!")
    print("How can I assist you with software installation?")

    while True:
        user_input = input("You: ")
        processed_input = process_input(user_input)

        # Check for exit command
        if processed_input == "exit":
            print("Goodbye!")
            break

        # Placeholder entity extraction (e.g., software name, operating system)
        software_name = "Example Software"
        operating_system = "Windows"  # Default to Windows for demonstration

        # Check user input for software name and operating system
        if "install" in processed_input:
            response = install_software(software_name, operating_system)
            print("Bot:", response)
        else:
            print("Bot: Sorry, I couldn't understand your request.")

# Run the chatbot
software_installation_chatbot()
```

```
Welcome to the Software Installation Chatbot!
How can I assist you with software installation?
You: I want to install software XYZ
Bot: Please follow these steps to install Example Software on Windows:
Step 1: Download the installer from the official website.
Step 2: Run the installer and follow the on-screen instructions.
Step 3: Complete the installation process.
If you encounter any issues, feel free to ask for assistance.
You: exit
Goodbye!
```

7A. Detecting Fake News

In [6]:
```
import re
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.pipeline import Pipeline
# Sample dataset of news articles (fake and real)
fake_news = ["This news article contains false information.",
 "The source of this news is unreliable and should not be trusted.",
 "The article claims outrageous things without providing evidence."]
real_news = ["This news article is based on verified facts and reliable sources.",
 "The information presented in this article has been confirmed by multiple sources.",
 "The news source has a history of producing accurate and trustworthy reporting."]
def preprocess_text(text):
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    text = text.lower()
    return text
all_news = fake_news + real_news
labels = [0] * len(fake_news) + [1] * len(real_news)
all_news_preprocessed = [preprocess_text(text) for text in all_news]
X_train, X_test, y_train, y_test = train_test_split(all_news_preprocessed, labels, test_size=0.2, random_state=42)
pipeline = Pipeline([
 ('tfidf', TfidfVectorizer()),
 ('clf', LogisticRegression())
])
pipeline.fit(X_train, y_train)
y_pred = pipeline.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.0
```

7B. Finding the most Viewed News Articles

In [7]:
```
news_articles = [
 {"title": "Breaking News: Earthquake Hits City A", "views": 1000},
 {"title": "New Study Reveals Benefits of Exercise", "views": 1500},
 {"title": "Government Announces Tax Reform Plan", "views": 800},
 {"title": "Tech Giant Launches New Smartphone", "views": 2000},
 {"title": "Local Team Wins Championship", "views": 1200}
]
# Sort the news articles based on view counts
most_viewed_articles = sorted(news_articles, key=lambda x: x["views"], reverse=True)
# Output the most viewed news articles
print("Top 3 Most Viewed News Articles:")
for idx, article in enumerate(most_viewed_articles[:3], 1):
    print(f"{idx}. {article['title']} - Views: {article['views']}")
```

```
Top 3 Most Viewed News Articles:
1. Tech Giant Launches New Smartphone - Views: 2000
2. New Study Reveals Benefits of Exercise - Views: 1500
3. Local Team Wins Championship - Views: 1200
```