ThreatSense

Team Members

Dinorah Garcia Vasquez

Elizabeth Cabrera

Hazel Ortega

Mustafa Eren

Idea of the Product

Introducing ThreatSense: Cybersecurity meets ML

- **ThreatSense Overview:** Web application leveraging AI to distinguish between malicious and safe emails and URLs.
- **Core Technology:** Powered by advanced machine learning algorithms.

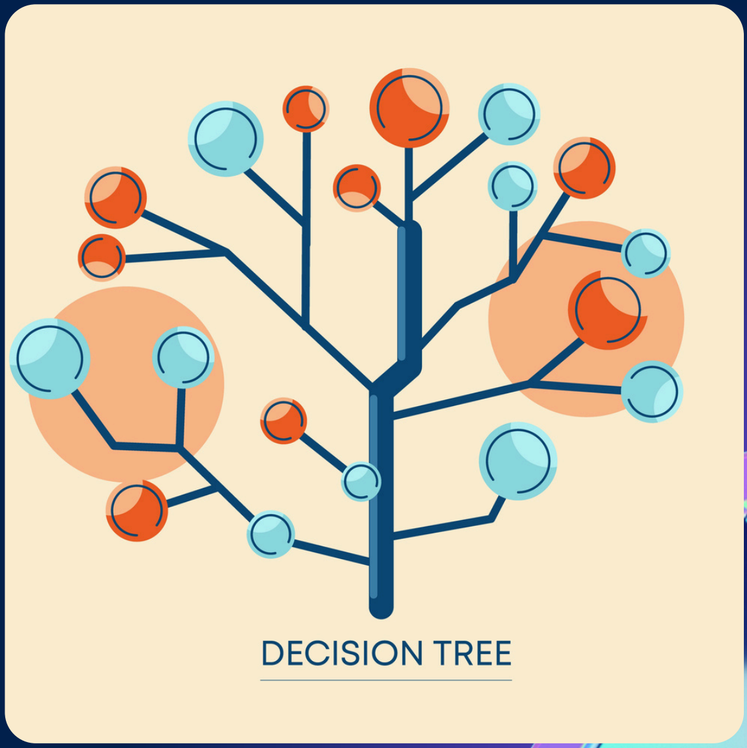
- **Purpose of AI Integration:** Enhances security analysts' decision-making. Provides support during incident response and reconnaissance, not to replace human expertise.
- **Functionality:** Acts as an independent tool to refine threat response processes.

Research Background

Algorithm & Comparison Previous Products

- **Random-Forest** is chosen for ThreatSense due to its effective classification, regression abilities, and high accuracy in decision-making.
- It builds **multiple decision trees** to enhance prediction reliability and **reduce over-fitting**, ideal for handling high dimensional data.

- Unlike other tools, ThreatSense integrates AI and ML deeply to **minimize human intervention** in threat detection and response.
- It aspires to autonomously **classify and accurately predict** the nature of emails and URLs, setting a new standard in cybersecurity tools.



DECISION TREE

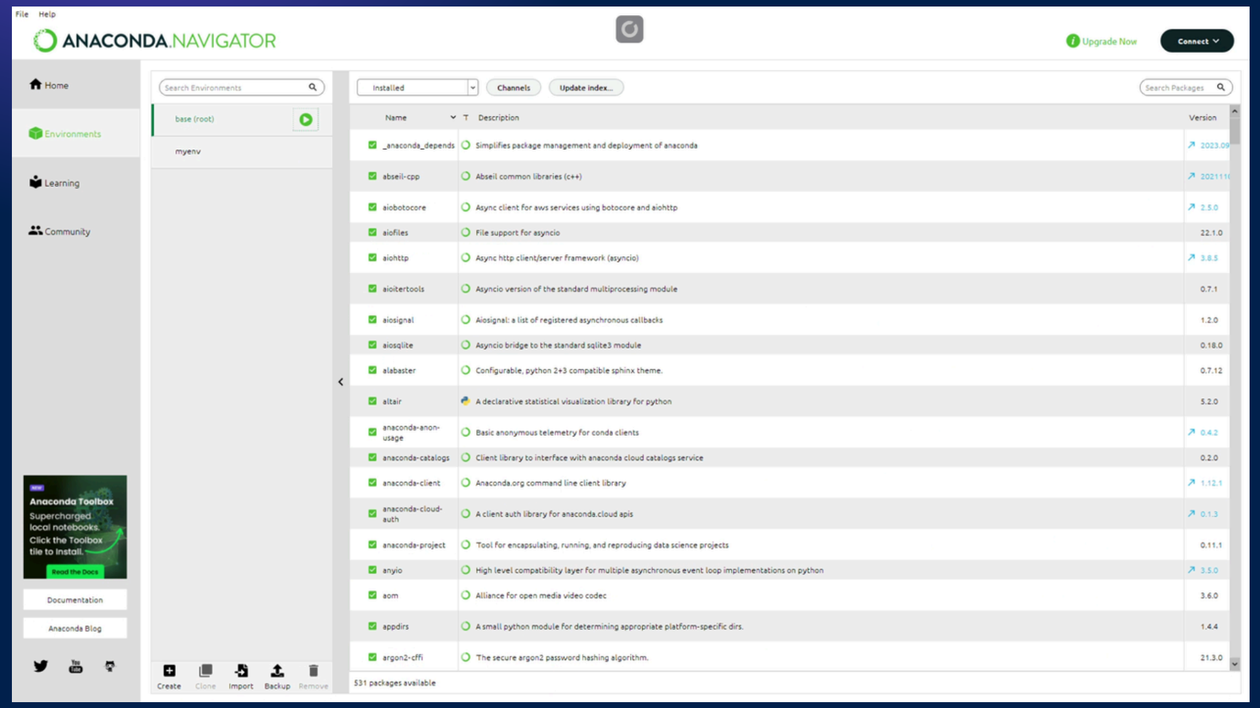
Technical Implementation and Environment of ThreatSense

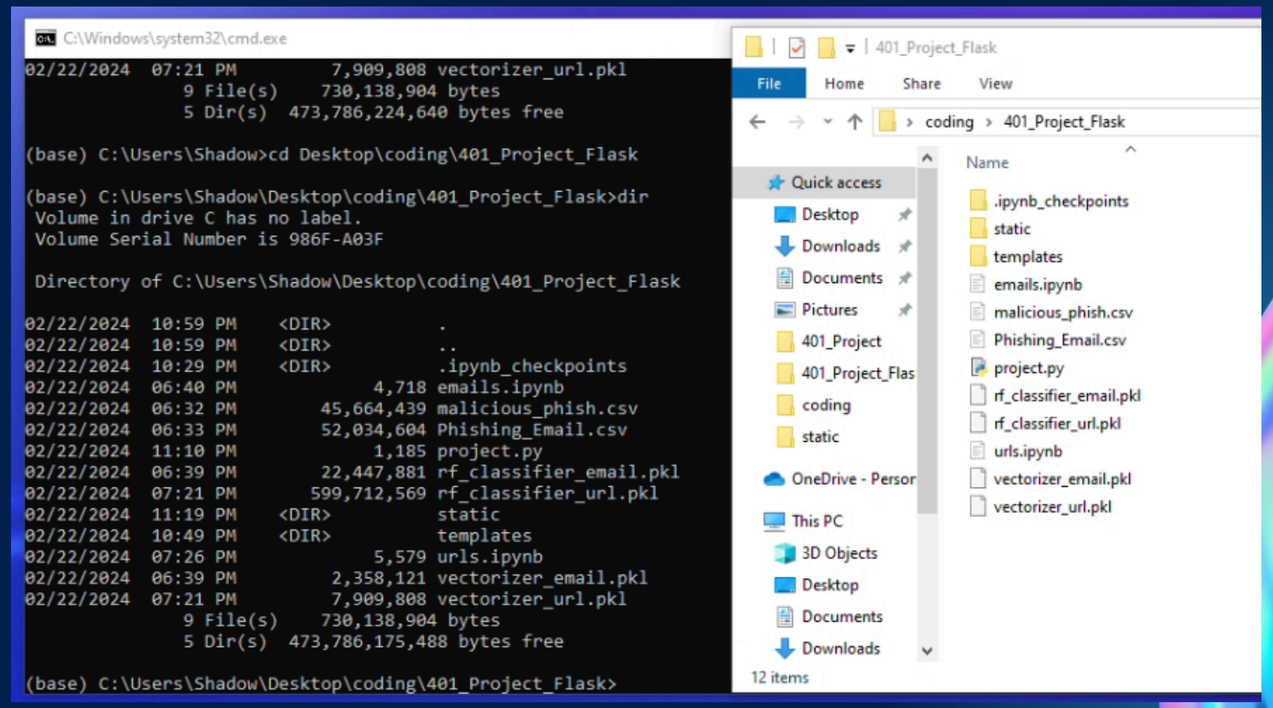
- ThreatSense uses AI and ML to classify URLs and emails, running in a **Windows testing environment** and not yet public.
- Utilizes **Anaconda for package management**, simplifying deployment and coding in a Windows environment.

- Organized in a single project directory, integrating **datasets, Jupyter Notebook, HTML, and CSS** for streamlined operation.

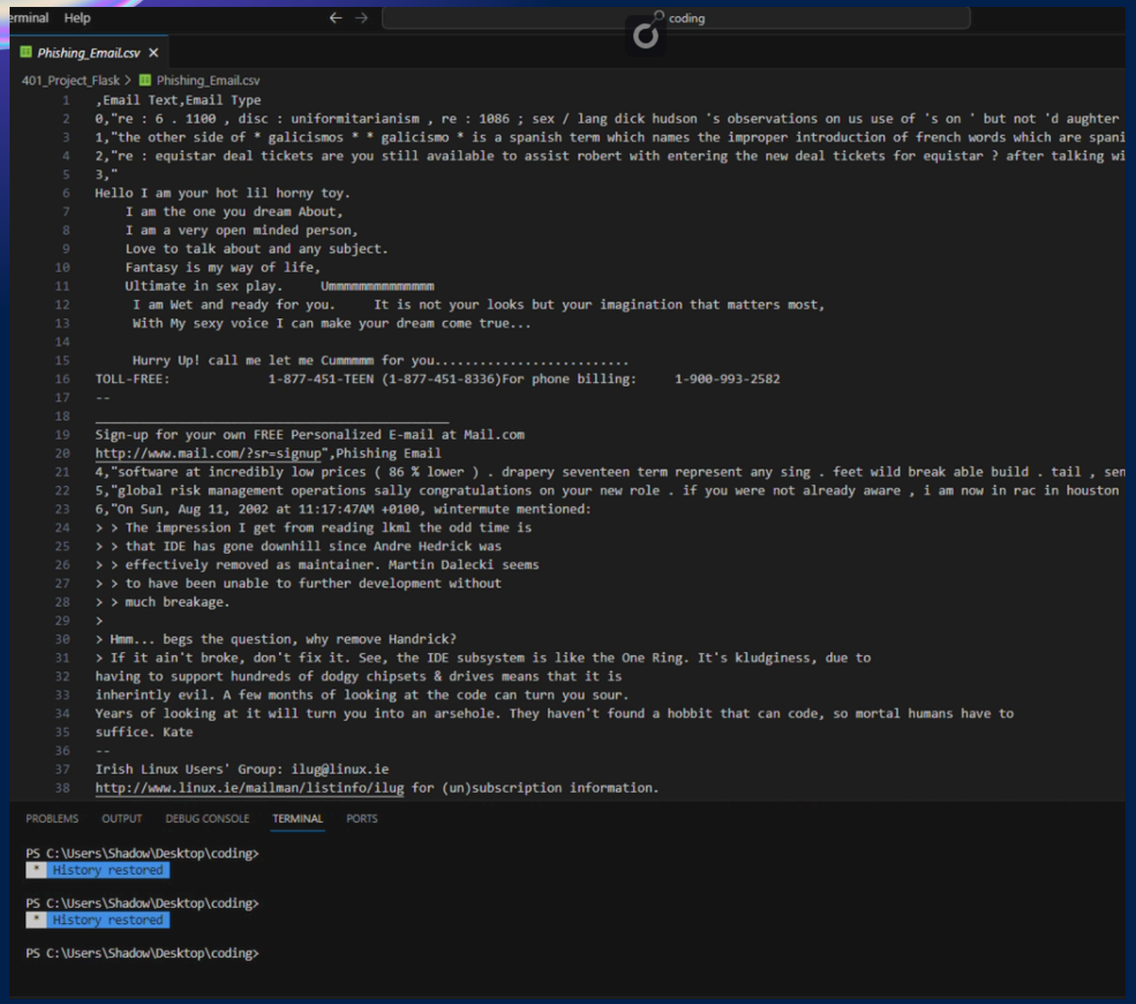
- Leverages Python and libraries like **sklearn and pandas** for efficient model training and application development.
- Hosted with **Flask**, a lightweight Python framework, enabling easy setup and rapid development of the web interface.

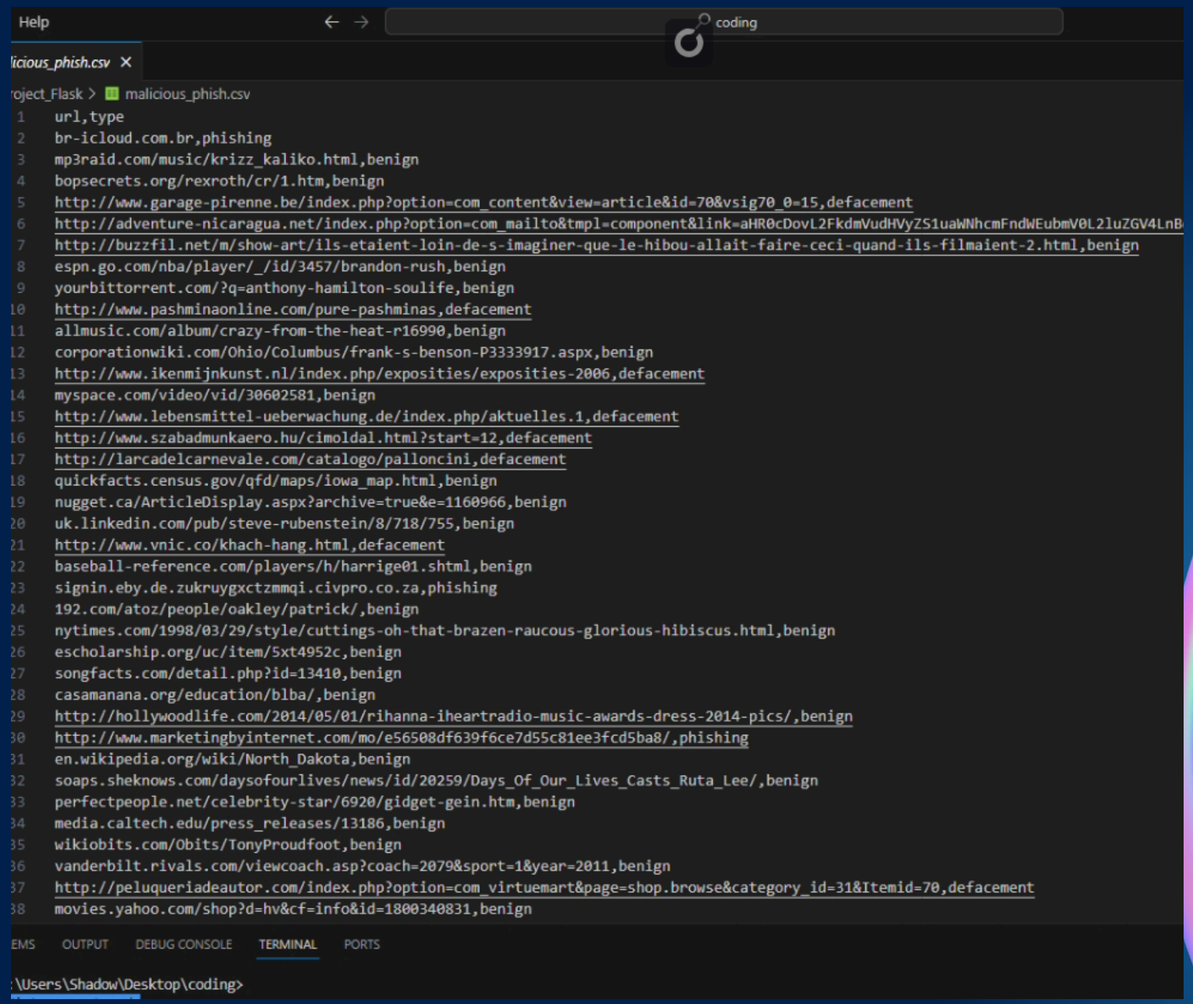
Anaconda Package Management & Directory Structure





URL and Email Datasets





Training the Model with Datasets

```
# Loading the dataset to the program
df = pd.read_csv('Phishing_Email.csv')

# Cleaning the null/NA values
df.dropna(inplace=True)

# Preprocessing the email content
def clean_text(text):
    # Removing the punctuations
    text = text.translate(str.maketrans('', '', string.punctuation))
    # Converting all the characters to lowercase for consistent text analysis
    text = text.lower()
    # Text is splitted into words and for processing purposes(counting frequency or finding patterns)
    words = re.findall(r'\b\w+\b', text)
    return ' '.join(words)

# Applying the clean text function for the related column in the dataset
df['Email Text'] = df['Email Text'].apply(clean_text)
```

```
# Data splitting for traning and testing
X_train, X_test, y_train, y_test = train_test_split(df['Email Text'], df['Email Type'], test_size=0.2, random_state=42)

# Converting the data into numerical values for vectorization that allows ML algorithms to quantitive analysis
vectorizer = TfidfVectorizer(max_features=5000)
X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)

# Training the model using RandomForest algorithm that uses 100 decision trees
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
rf_classifier.fit(X_train_tfidf, y_train)

# Checking the accuracy of the model
y_pred = rf_classifier.predict(X_test_tfidf)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

HTML File

```
</head>
<body>  <!-- Setting up the webpage with two user input box and buttons. This part also provides what each button will do.-->
    <div id="container">
        <h1></h1>
        <div class="form-container">
            <h2>Email Threat Detector</h2>
            <form action="/predict_email" method="post">
                <textarea name="email_text" rows="4" cols="50" placeholder="Input email text here">{{ email_text|default('') }}</textarea><br>
                <input type="submit" value="Analyze">
                {% if email_prediction %}
                    <p>Email Prediction: {{ email_prediction }}</p>
                    <p><strong>Input:</strong> {{ email_text }}</p>
                {% endif %}
            </form>

            <h2>URL Threat Detector</h2>
            <form action="/predict_url" method="post">
                <input type="text" name="url_text" placeholder="Input URL here" value="{{ url_text|default('') }}"><br>
                <input type="submit" value="Analyze">
                {% if url_prediction %}
                    <p>URL Prediction: {{ url_prediction }}</p>
                    <p><strong>Input:</strong> {{ url_text }}</p>
                {% endif %}
            </form>
        </div>
```

Main Python File

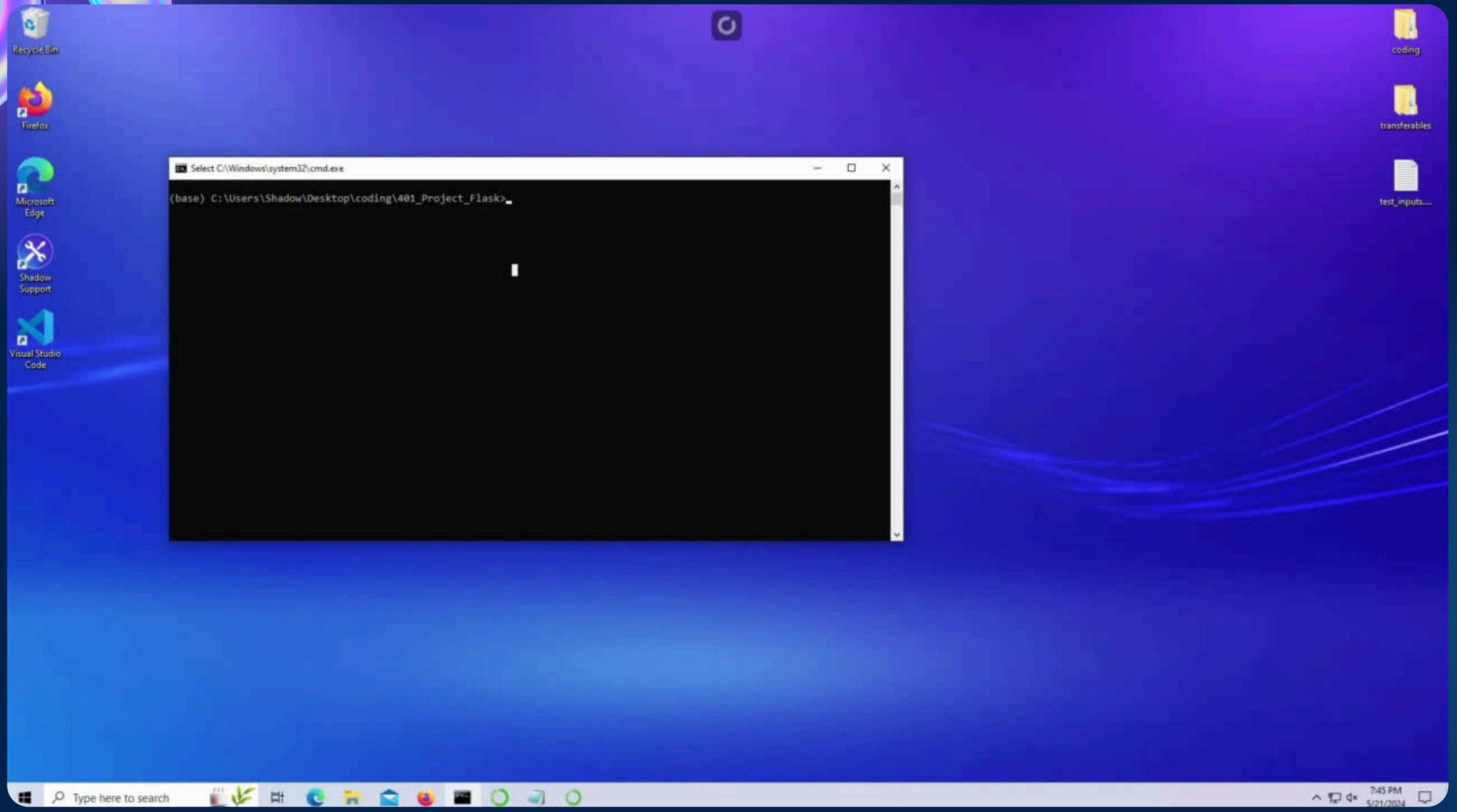
```python
# Route for the homepage
@app.route('/')
def home():
    return render_template('index.html') #index.html holds the actual website design

# Route for email prediction
@app.route('/predict_email', methods=['POST'])
def predict_email():
    email_text = request.form['email_text']
    email_text_vectorized = email_vectorizer.transform([email_text])
    email_prediction = email_model.predict(email_text_vectorized)
    # Truncating the email_text to 100 characters plus 10 dots for more pleasant visual
    truncated_email_text = (email_text[:100] + '..........') if len(email_text) > 100 else email_text
    return render_template('index.html', email_prediction=email_prediction[0], email_text=truncated_email_text)
```

```python
# Route for URL prediction
@app.route('/predict_url', methods=['POST'])
def predict_url():
    url_text = request.form['url_text']
    if is_whitelisted(url_text):
        url_prediction = 'benign'
    else:
        url_text_vectorized = url_vectorizer.transform([url_text])
        url_prediction = url_model.predict(url_text_vectorized)
        url_prediction = url_prediction[0]  # Getting decision from prediction array
    return render_template('index.html', url_prediction=url_prediction, url_text=url_text)

if __name__ == '__main__':
    app.run(debug=True)
```

Threatsense – Live Demo

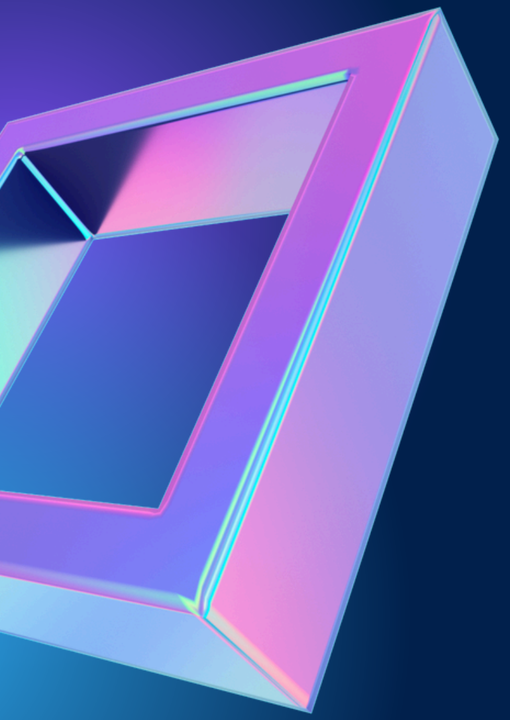


From Challenges to Solutions: Key Takeaways

- **Data Collection Challenges:** Initial difficulty in obtaining high-quality, up-to-date datasets for AI training. Shifted strategy to collect email samples directly from John Jay College, supplemented by Kaggle datasets when direct collection was not possible.
- **Dataset Handling:** Faced challenges with cleaning and preparing URLs in the datasets for effective use.

- **Advancements in Phishing Detection:** Explored and tried to implement Natural Language Processing (NLP) to enhance phishing detection capabilities. NLP models tailored specifically to understand and detect nuances in phishing attempts.
- **Integration Setbacks:** Encountered resource limitations preventing the integration of the fine-tuned NLP model into the website.

- **Exploring Solutions and Future Possibilities:** Considered using cloud resources like AWS SageMaker to overcome resource constraints and enhance our AI capabilities. Ongoing efforts to develop more robust solutions with adequate resources.



Thank You!

https://www.canva.com/design/DAGF5GRLKG8/inzJSx1SOOz9OlMeI5i7Ig/view?utm_content=DAGF5GRLKG8&utm_campaign=designshare&utm_medium=link&utm_source=editor