1. Data Preprocessing

```python
import pandas as pd

df=pd.read_csv("../data/initial_urls.csv")
df2=pd.read_csv("../data/additional_urls.csv")

def initial_read(df):...

df.dropna(inplace=True)
df2.drop(columns = ['Unnamed: 0', 'label'], inplace=True)

df['label']=df['label'].astype(int)

df.rename(columns={"domain": "url", "label": "phishing"},inplace=True)
df2.rename(columns={"result": "phishing"},inplace=True)

df['url']= 'https://' + df['url'].astype(str)

df_final = pd.concat([df,df2])
df_final .drop_duplicates(inplace=True)

df_final.to_csv('../data/final_urls.csv', index=False)
print(df.columns)
print(df2.columns)
```

2. Feature Extraction

```python
def getEntropy(url):
    url = url.lower()
    probs = [url.count(c) / len(url) for c in set(url)]
    entropy = -sum([p * log(p) / log(2.0) for p in probs])
    return entropy

def hasLogin(url):
    return int('login' in url.lower())

def redirection(url):
    pos = url.rfind('//')
    if pos > 6:
        if pos > 7:
            return 1
        else:
            return 0
    else:
        return 0

def lenClassify(url):
    if len(url) < 54:
        length = 0
    else:
        length = 1
    return length

def haveAtSign(url):
    if "@" in url:
        at = 1
    else:
        at = 0
    return at
```

```python
def getDepth(url):
    s = urlparse(url).path.split('/')
    depth = 0
    for j in range(len(s)):
        if len(s[j]) != 0:
            depth = depth+1
    return depth

def tinyURL(url):
    match=re.search(shortening_services,url)
    if match:
        return 1
    else:
        return 0

def isDomainIp(domain):
    domain = domain.split(':')
    pattern = r'^(?:[0-9]{1,3}\.){3}[0-9]{1,3}$|^ \
              (?:[a-f0-9]{1,4}:){7}[a-f0-9]{1,4}$'
    match = re.match(pattern, domain[0])
    if match is not None:
        return 1
    else:
        return 0

def prefixSuffix(domain):
    if '-' in domain:
        return 1
    else:
        return 0
```

```
needed_cols = ['url', 'domain', 'path', 'query', 'fragment']
for col in needed_cols:
    df[f'{col}_length']=df[col].str.len()
    df[f'qty_dot_{col}'] = df[[col]].applymap(lambda x: str.count(x, '.'))
    df[f'qty_hyphen_{col}'] = df[[col]].applymap(lambda x: str.count(x, '-'))
    df[f'qty_slash_{col}'] = df[[col]].applymap(lambda x: str.count(x, '/'))
    df[f'qty_questionmark_{col}'] = df[[col]].applymap(lambda x: str.count(x, '?'))
    df[f'qty_equal_{col}'] = df[[col]].applymap(lambda x: str.count(x, '='))
    df[f'qty_at_{col}'] = df[[col]].applymap(lambda x: str.count(x, '@'))
    df[f'qty_and_{col}'] = df[[col]].applymap(lambda x: str.count(x, '&'))
    df[f'qty_exclamation_{col}'] = df[[col]].applymap(lambda x: str.count(x, '!'))
    df[f'qty_space_{col}'] = df[[col]].applymap(lambda x: str.count(x, ' '))
    df[f'qty_tilde_{col}'] = df[[col]].applymap(lambda x: str.count(x, '~'))
    df[f'qty_comma_{col}'] = df[[col]].applymap(lambda x: str.count(x, ','))
    df[f'qty_plus_{col}'] = df[[col]].applymap(lambda x: str.count(x, '+'))
    df[f'qty_asterisk_{col}'] = df[[col]].applymap(lambda x: str.count(x, '*'))
    df[f'qty_hashtag_{col}'] = df[[col]].applymap(lambda x: str.count(x, '#'))
    df[f'qty_dollar_{col}'] = df[[col]].applymap(lambda x: str.count(x, '$'))
    df[f'qty_percent_{col}'] = df[[col]].applymap(lambda x: str.count(x, '%'))

col_in_question = ['qty_slash_domain', 'qty_questionmark_domain','qty_equal_domain', 'qty_at_domain', 'qty_and_domain',
'qty_exclamation_domain', 'qty_space_domain', 'qty_tilde_domain','qty_comma_domain', 'qty_plus_domain',
'qty_asterisk_domain','qty_hashtag_domain', 'qty_dollar_domain', 'qty_percent_domain', 'qty_questionmark_path',
'qty_hashtag_path', 'qty_hashtag_query', 'qty_at_fragment','qty_tilde_fragment', 'qty_plus_fragment']

df.drop(columns = col_in_question, inplace=True)
```

3. Pycaret Comparison

```
import pandas as pd
from pycaret.classification import *
import time

start_time = time.time()

df = pd.read_csv('/content/drive/MyDrive/ML-Phishing Detection Project/data/url_features.csv')
data = df.drop(columns=['url','protocol','domain','path','query','fragment'])

s = setup(data, target = 'phishing', session_id = 123)
best = compare_models()

print("\n--- pycaret check completed in %s seconds ---" % (time.time() - start_time))
```

4. Model Evaluation

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
import time
start_time = time.time()
import warnings
warnings.filterwarnings("ignore")

df = pd.read_csv("../data/url_features.csv")

x = df.drop(columns=['url','protocol','domain','path','query','fragment','phishing'])
y = df['phishing']
X_train, X_test, y_train, y_test = train_test_split(x,y,random_state = 42, stratify = y )

clf = LogisticRegression(penalty="l2",C=10,max_iter=1000, random_state=42)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

acc = accuracy_score(y_test, y_pred)*100
train_accuracy = clf.score(X_train, y_train)*100
test_accuracy = clf.score(X_test, y_test)*100

print(f"LogisticRegression Accuracy: {acc}")
print("Training accuracy:", train_accuracy)
print("Test accuracy:", test_accuracy)

print("\n--- Model Evaluation ended in %s seconds ---" % (time.time() - start_time))
```

5. Model Implementation

```
app.post('/api/url', (req, res) => {
    const data = req.body.url;
    try {
        parsedUrl = new URL(data);
        const dataStr = JSON.stringify(data);
        const pythonProcess = spawn('python', ['./predictor.py']);
        pythonProcess.stdin.write(dataStr);
        pythonProcess.stdin.end();
        let outputData = '';
        pythonProcess.stdout.on('data', (data) => {
            outputData += data;
        });
        pythonProcess.stderr.on('data', (data) => {
            res.status(400).json(error)
            console.error(`stderr: ${data}`);
        });
        pythonProcess.on('error', (error) => {
            res.status(400).json(error)
            console.error(`Python process error: ${error}`);
        });
        pythonProcess.on('close', (code) => {
        if (code !== 0) {
            res.status(505).json(`Python process exited with code ${code}`)
            console.error(`Python process exited with code ${code}`);
        } else {
            const result = JSON.parse(outputData);
            res.status(200).json(result.phishing_value)
        }
        });
    } catch (err) {
        res.status(400).json(err)
    }
 })
```

app.js

```
with open('./model/model.pkl', 'rb') as file: #LOAD MODEL
    model = pickle.load(file)
with open(filename, 'w', newline='') as file:
    df = pd.read_csv(filename,names=['url'])
    url=[]
    data = json.loads(sys.stdin.read()) # RECEIVE INPUT
    url.append(data)
    df["url"]=url
    urls = [url for url in df['url']]
    df['protocol'],df['domain'],df['path'],df['query'],df['fragment'] = \
                        zip(*[urllib.parse.urlsplit(x) for x in urls])
    shortening_services = r"bit\.ly|goo\.gl|shorte\.st|go2l\.ink|x\.co|ow\ \...
    # FEATURE EXTRACTION
    def getEntropy(url):...
    def hasLogin(url):...
    def redirection(url):...
    def lenClassify(url):...
    def haveAtSign(url):...
    def getDepth(url):...
    def tinyURL(url):...
    def isDomainIp(domain):...
    def prefixSuffix(domain):...
    def get_features(df):...
    get_features(df)
    df.to_csv("./dataTest/test_features.csv",index=False)
    file.close()

x_test = df.drop(columns=['url','protocol','domain','path','query','fragment'])
y_pred = model.predict(x_test) #PREDICT
result = {'phishing_value': str(y_pred[0])}
sys.stdout.write(json.dumps(result)) #SEND OUTPUT
```

predictor.py