1) PAGERANK:

```
import networkx as nx

# Create a directed graph
G = nx.DiGraph()

# Add edges to the graph
G.add_edges_from([(1, 2), (1, 3), (2, 1), (3, 2)])

# Calculate PageRank
pagerank = nx.pagerank(G)

# Print the PageRank scores
for node, score in pagerank.items():
    print(f"Node {node}: PageRank score = {score}")
```

2) JACCARD'S COEFFICIENT:

```
import networkx as nx

# Create a graph
G = nx.Graph()

# Add edges to the graph
G.add_edges_from([(1, 2), (1, 3), (2, 4), (3, 4), (4, 5)])

# Calculate the Jaccard coefficient for each pair of nodes
jaccard_coefficient = nx.jaccard_coefficient(G)

# Print the Jaccard coefficient for each pair of nodes
for u, v, coef in jaccard_coefficient:
    print(f"Nodes ({u}, {v}): Jaccard coefficient = {coef}")
```

3) LINK PREDICTION USING NODE2VEC:

```
import networkx as nx
from node2vec import Node2Vec
from sklearn.metrics.pairwise import cosine_similarity

# Create a graph
G = nx.Graph()

# Add edges to the graph
G.add_edges_from([(1, 2), (1, 3), (2, 4), (3, 4), (4, 5)])
```

```
# Generate Node2Vec embeddings
node2vec = Node2Vec(G, dimensions=64, walk_length=30, num_walks=200)

# Learn embeddings
model = node2vec.fit(window=10, min_count=1, batch_words=4)

# Get embeddings for all nodes
embeddings = {node: model.wv[str(node)] for node in G.nodes()}

# Perform link prediction using cosine similarity
link_predictions = []
for edge in nx.non_edges(G):
    source, target = edge
    source_embedding = embeddings[source]
    target_embedding = embeddings[target]
    similarity = cosine_similarity([source_embedding],
[target_embedding])[0][0]
    link_predictions.append((source, target, similarity))

# Sort the link predictions in descending order of similarity
link_predictions = sorted(link_predictions, key=lambda x: x[2],
reverse=True)

# Print the top recommended links
top_links = link_predictions[:5]
for source, target, similarity in top_links:
    print(f"Recommended Link: ({source}, {target}) (Similarity:
{similarity})")
```

4a) HYPERLINKS:

```
import requests
from bs4 import BeautifulSoup

# Specify the URL of the webpage
url = "https://en.wikipedia.org/wiki/OpenAI"

# Send a GET request to the webpage
response = requests.get(url)

# Create a BeautifulSoup object to parse the HTML content
soup = BeautifulSoup(response.content, "html.parser")

# Find all anchor tags (<a>) in the parsed HTML
anchors = soup.find_all("a")
```

```
# Extract the href attribute from each anchor tag
hyperlinks = [anchor.get("href") for anchor in anchors]

# Print the extracted hyperlinks
for hyperlink in hyperlinks:
    print(hyperlink)
```

4b) TITLES:

```
import requests
from bs4 import BeautifulSoup

# Specify the URL of the Wikipedia webpage
url = "https://en.wikipedia.org/wiki/OpenAI"

# Send a GET request to the webpage
response = requests.get(url)

# Create a BeautifulSoup object to parse the HTML content
soup = BeautifulSoup(response.content, "html.parser")

# Find the main title of the Wikipedia page
title = soup.find("h1", id="firstHeading").text

# Find all other titles in the page (subheadings, sections, etc.)
titles = [title.text for title in soup.find_all(["h2", "h3", "h4", "h5",
"h6"])]

# Print the main title and other titles
print("Main Title:", title)
print("Other Titles:")
for title in titles:
    print(title)
```

5) STACK OVERFLOW:

```
import requests

def search_stack_overflow(query):
    base_url = "https://api.stackexchange.com/2.3/search"
    params = {
        "order": "desc",
        "sort": "relevance",
        "site": "stackoverflow",
        "intitle": query
    }
    response = requests.get(base_url, params=params)
    data = response.json()

    if "items" in data:
        for item in data["items"]:
            title = item["title"]
            link = item["link"]
            print(f"Title: {title}\nLink: {link}\n")
    else:
        print("No relevant results found.")

# Example usage
if __name__ == "__main__":
    query = "python error syntax"
    search_stack_overflow(query)
```

6) PYTRENDS:

```
from pytrends.request import TrendReq
import pandas as pd

# Connect to Google
pytrends = TrendReq(hl='en-US', tz=360)

# Build payload
keyword = 'Python programming'
pytrends.build_payload([keyword])

# Get interest over time data
interest_over_time_df = pytrends.interest_over_time()

# Retrieve historical interest data
historical_interest_data = pd.DataFrame()
for year in range(2010, 2022):
    pytrends.build_payload([keyword], timeframe=f'{year}-01-01
{year}-12-31')
    historical_interest_data = pd.concat([historical_interest_data,
pytrends.interest_over_time()])

# Interest by Region
interest_by_region_df = pytrends.interest_by_region()

# Top Charts
top_charts_df = pytrends.top_charts(2021, hl='en-US', tz=360,
geo='GLOBAL')
```

```python
# Related Queries
related_queries_dict = pytrends.related_queries()

# Keyword Suggestion
keyword_suggestions_list = pytrends.suggestions(keyword)

# Print the results
print("Interest Over Time:")
print(interest_over_time_df.head())

print("\nHistorical Interest Data:")
print(historical_interest_data)

print("\nInterest by Region:")
print(interest_by_region_df.head())

print("\nTop Charts:")
print(top_charts_df.head())

print("\nRelated Queries:")
print(related_queries_dict[keyword])

print("\nKeyword Suggestion:")
print(keyword_suggestions_list)
```