```
# First, backup your current file
#cp scripts/era5_analysis_01.py scripts/era5_analysis_01.py.backup

# Create a corrected version
#cat > scripts/era5_analysis_01.py << 'EOF'
#!/usr/bin/env python3
"""
ERA5 Analysis for Kariba Wind Study
Downloads and processes ERA5 data for Lake Kariba region
"""
import os
import sys
import time
import json
from pathlib import Path
import numpy as np
import pandas as pd
import xarray as xr
import matplotlib.pyplot as plt
from datetime import datetime, timedelta
import logging

# Add project root to path
project_root = Path('/home/chawas/deployed/charara_01')
sys.path.append(str(project_root))

class ERA5Downloader:
"""Download ERA5 data for Kariba region"""
def __init__(self, config_file=None):
if config_file is None:
# Use default config path
config_file = project_root / 'config' / 'kariba_config.yaml'
# Load configuration
try:
import yaml
with open(config_file, 'r') as f:
self.config = yaml.safe_load(f)
except FileNotFoundError:
print(f"Config file not found: {config_file}")
print("Using default configuration...")
self.config = self._default_config()
# Setup paths
self.project_dir = project_root
self.data_dir = self.project_dir / 'data' / 'era5'
self.data_dir.mkdir(parents=True, exist_ok=True)
# Setup logger
self.logger = self._setup_logger()
# Test CDS API
self._test_cdsapi()
def _default_config(self):
```

```
"""Default configuration if file doesn't exist"""
return {
'project': {
'start_year': 1990,
'end_year': 2020
},
'lake': {
'charara_point': [-16.53, 28.83],
'coordinates': {
'north': -15.75,
'south': -17.25,
'west': 27.25,
'east': 29.00
}
},
'era5': {
'variables': [
'10m_u_component_of_wind',
'10m_v_component_of_wind',
'2m_temperature'
],
'region': [-10.0, 26.0, -18.0, 30.0]
}
}
def _setup_logger(self):
"""Setup logging"""
log_dir = self.project_dir / 'logs'
log_dir.mkdir(exist_ok=True)
logging.basicConfig(
level=logging.INFO,
format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
handlers=[
logging.FileHandler(log_dir / 'era5_download.log'),
logging.StreamHandler()
]
)
return logging.getLogger(__name__)
def _test_cdsapi(self):
"""Test if CDS API is available"""
self.logger.info("Testing CDS API...")
try:
import cdsapi
self.logger.info("✓ CDS API available")
return True
except ImportError:
self.logger.error("✗ CDS API not installed. Install with: pip install cdsapi")
return False
except Exception as e:
self.logger.error(f"✗ CDS API error: {e}")
return False
```

```
def download_data(self, start_year=None, end_year=None, force=False):
"""Download ERA5 data"""
if not self._test_cdsapi():
self.logger.error("Cannot download without CDS API")
return []
import cdsapi
# Use config years if not specified
if start_year is None:
start_year = self.config['project']['start_year']
if end_year is None:
end_year = self.config['project']['end_year']
self.logger.info(f"Downloading ERA5 data for {start_year}-{end_year}")
c = cdsapi.Client()
downloaded_files = []
for year in range(start_year, end_year + 1):
self.logger.info(f"Processing year {year}")
for month in range(1, 13):
filename = self.data_dir / f'era5_kariba_{year}_{month:02d}.nc'
# Skip if file exists and not forcing
if filename.exists() and not force:
self.logger.info(f" Month {month:02d}: File exists")
downloaded_files.append(filename)
continue
self.logger.info(f" Month {month:02d}: Downloading...")
try:
# Prepare request
request = {
'product_type': 'reanalysis',
'variable': self.config['era5']['variables'],
'year': str(year),
'month': f'{month:02d}',
'day': [f'{d:02d}' for d in range(1, 32)],
'time': [f'{h:02d}:00' for h in range(24)],
'area': self.config['era5']['region'],
'format': 'netcdf',
}
# Download
c.retrieve('reanalysis-era5-single-levels', request, str(filename))
self.logger.info(f" Month {month:02d}: Downloaded")
downloaded_files.append(filename)
# Be nice to the API
time.sleep(2)
except Exception as e:
self.logger.error(f" Month {month:02d}: Error - {e}")
continue
self.logger.info(f"Download complete. {len(downloaded_files)} files downloaded.")
return downloaded_files
def process_downloaded_data(self, era5_files=None):
"""Process downloaded ERA5 data"""
self.logger.info("Processing ERA5 data...")
```

```
if era5_files is None:
# Find all ERA5 files
era5_files = sorted(self.data_dir.glob('era5_kariba_*.nc'))
if not era5_files:
self.logger.error("No ERA5 files found")
return None
self.logger.info(f"Found {len(era5_files)} ERA5 files")
# Process in chunks to avoid memory issues
processed_dir = self.project_dir / 'data' / 'era5_processed'
processed_dir.mkdir(parents=True, exist_ok=True)
# Process each year separately
yearly_datasets = []
# Group files by year
files_by_year = {}
for file in era5_files:
year = int(file.stem.split('_')[2])
if year not in files_by_year:
files_by_year[year] = []
files_by_year[year].append(file)
for year, year_files in files_by_year.items():
self.logger.info(f"Processing year {year}...")
try:
# Open and combine monthly files for this year
ds_list = []
for file in sorted(year_files):
ds = xr.open_dataset(file)
ds_list.append(ds)
if ds_list:
ds_year = xr.concat(ds_list, dim='time')
# Calculate wind speed and direction
if 'u10' in ds_year and 'v10' in ds_year:
ds_year['wind_speed_10m'] = np.sqrt(
ds_year['u10']**2 + ds_year['v10']**2
)
ds_year['wind_dir_10m'] = (
270 - np.arctan2(ds_year['v10'], ds_year['u10']) * 180/np.pi
) % 360
# Convert temperature to Celsius
if 't2m' in ds_year:
ds_year['temperature_2m'] = ds_year['t2m'] - 273.15
# Save yearly dataset
yearly_file = processed_dir / f'era5_kariba_{year}.nc'
ds_year.to_netcdf(yearly_file)
yearly_datasets.append(yearly_file)
self.logger.info(f" Saved: {yearly_file.name}")
except Exception as e:
self.logger.error(f" Error processing {year}: {e}")
continue
# Combine all years if we have multiple
if len(yearly_datasets) > 1:
```

```
self.logger.info("Combining all years...")
ds_list = []
for file in yearly_datasets:
ds = xr.open_dataset(file)
ds_list.append(ds)
ds_combined = xr.concat(ds_list, dim='time')
# Save combined dataset
combined_file = processed_dir / 'era5_kariba_1990_2020.nc'
ds_combined.to_netcdf(combined_file)
self.logger.info(f"Saved combined dataset: {combined_file}")
return ds_combined
elif yearly_datasets:
# Only one year
ds = xr.open_dataset(yearly_datasets[0])
return ds
else:
self.logger.error("No data processed successfully")
return None
def extract_charara_data(self, ds_era5):
"""Extract data at Charara point"""
self.logger.info("Extracting Charara point data...")
charara_lat, charara_lon = self.config['lake']['charara_point']
# Find nearest grid point
lat = ds_era5.latitude.values
lon = ds_era5.longitude.values
lat_idx = np.argmin(np.abs(lat - charara_lat))
lon_idx = np.argmin(np.abs(lon - charara_lon))
nearest_lat = lat[lat_idx]
nearest_lon = lon[lon_idx]
self.logger.info(f"Charara coordinates: {charara_lat}, {charara_lon}")
self.logger.info(f"Nearest ERA5 grid: {nearest_lat:.3f}, {nearest_lon:.3f}")
# Calculate distance
R = 6371 # Earth radius in km
lat1, lon1, lat2, lon2 = map(np.radians, [charara_lat, charara_lon, nearest_lat, nearest_lon])
dlat = lat2 - lat1
dlon = lon2 - lon1
a = np.sin(dlat/2)**2 + np.cos(lat1) * np.cos(lat2) * np.sin(dlon/2)**2
c = 2 * np.arcsin(np.sqrt(a))
distance = R * c
self.logger.info(f"Grid distance: {distance:.1f} km")
# Extract timeseries
charara_data = {
'time': ds_era5.time.values,
'wind_speed_10m': ds_era5['wind_speed_10m'][:, lat_idx, lon_idx].values,
'wind_dir_10m': ds_era5['wind_dir_10m'][:, lat_idx, lon_idx].values,
'temperature_2m': ds_era5['temperature_2m'][:, lat_idx, lon_idx].values,
'u10': ds_era5['u10'][:, lat_idx, lon_idx].values,
'v10': ds_era5['v10'][:, lat_idx, lon_idx].values,
}
# Create DataFrame
```

```
df = pd.DataFrame(charara_data)
df['time'] = pd.to_datetime(df['time'])
df.set_index('time', inplace=True)
# Save to CSV
output_dir = self.project_dir / 'outputs'
output_dir.mkdir(parents=True, exist_ok=True)
csv_file = output_dir / 'charara_era5_timeseries.csv'
df.to_csv(csv_file)
self.logger.info(f"Saved Charara timeseries: {csv_file}")
# Calculate basic statistics
self._calculate_statistics(df)
return df
def _calculate_statistics(self, df):
"""Calculate basic wind statistics"""
stats = {
'Mean wind speed': f"{df['wind_speed_10m'].mean():.2f} m/s",
'Median wind speed': f"{df['wind_speed_10m'].median():.2f} m/s",
'Std wind speed': f"{df['wind_speed_10m'].std():.2f} m/s",
'Max wind speed': f"{df['wind_speed_10m'].max():.2f} m/s",
'Mean temperature': f"{df['temperature_2m'].mean():.2f} °C",
}
self.logger.info("Charara Wind Statistics:")
for key, value in stats.items():
self.logger.info(f" {key}: {value}")
# Save statistics to file
stats_file = self.project_dir / 'outputs' / 'charara_statistics.txt'
with open(stats_file, 'w') as f:
f.write("CHARARA WIND STATISTICS (ERA5)\n")
f.write("=" * 40 + "\n\n")
for key, value in stats.items():
f.write(f"{key}: {value}\n")
self.logger.info(f"Statistics saved: {stats_file}")
def create_basic_plots(self, df):
"""Create basic wind analysis plots"""
self.logger.info("Creating basic plots...")
plots_dir = self.project_dir / 'outputs' / 'plots'
plots_dir.mkdir(parents=True, exist_ok=True)
# 1. Time series plot
plt.figure(figsize=(12, 8))
plt.subplot(3, 1, 1)
plt.plot(df.index, df['wind_speed_10m'], 'b-', alpha=0.7, linewidth=0.5)
plt.ylabel('Wind Speed (m/s)')
plt.title('Charara Wind Speed - ERA5')
plt.grid(True, alpha=0.3)
plt.subplot(3, 1, 2)
plt.plot(df.index, df['temperature_2m'], 'r-', alpha=0.7, linewidth=0.5)
plt.ylabel('Temperature (°C)')
plt.grid(True, alpha=0.3)
plt.subplot(3, 1, 3)
plt.plot(df.index, df['wind_dir_10m'], 'g-', alpha=0.7, linewidth=0.5)
plt.ylabel('Wind Direction (°)')
plt.xlabel('Time')
plt.grid(True, alpha=0.3)
plt.tight_layout()
ts_plot = plots_dir / 'charara_wind_timeseries.png'
plt.savefig(ts_plot, dpi=150, bbox_inches='tight')
plt.close()
self.logger.info(f"Created: {ts_plot}")
# 2. Diurnal cycle
df['hour'] = df.index.hour
diurnal_speed = df.groupby('hour')['wind_speed_10m'].mean()
diurnal_temp = df.groupby('hour')['temperature_2m'].mean()
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 8))
ax1.plot(diurnal_speed.index, diurnal_speed.values, 'b-o', linewidth=2)
ax1.set_xlabel('Hour of Day')
ax1.set_ylabel('Wind Speed (m/s)')
ax1.set_title('Diurnal Cycle - Wind Speed')
ax1.grid(True, alpha=0.3)
ax1.set_xticks(range(0, 24, 3))
ax2.plot(diurnal_temp.index, diurnal_temp.values, 'r-o', linewidth=2)
ax2.set_xlabel('Hour of Day')
ax2.set_ylabel('Temperature (°C)')
ax2.set_title('Diurnal Cycle - Temperature')
ax2.grid(True, alpha=0.3)
ax2.set_xticks(range(0, 24, 3))
plt.tight_layout()
diurnal_plot = plots_dir / 'charara_diurnal_cycle.png'
plt.savefig(diurnal_plot, dpi=150, bbox_inches='tight')
plt.close()
self.logger.info(f"Created: {diurnal_plot}")
# 3. Wind rose (if windrose package is available)
try:
from windrose import WindroseAxes
fig = plt.figure(figsize=(10, 8))
ax = WindroseAxes.from_ax(fig=fig)
ax.bar(df['wind_dir_10m'], df['wind_speed_10m'],
normed=True, opening=0.8, edgecolor='white',
bins=[0, 2, 5, 8, 12, 20])
ax.set_legend(title='Wind Speed (m/s)', loc='lower left')
ax.set_title('Charara Wind Rose - ERA5')
windrose_plot = plots_dir / 'charara_windrose.png'
plt.savefig(windrose_plot, dpi=150, bbox_inches='tight')
plt.close()
self.logger.info(f"Created: {windrose_plot}")
except ImportError:
self.logger.warning("Windrose library not installed. Install with: pip install windrose")
self.logger.info("Plot creation complete!")

def main():
"""Main function to run ERA5 analysis"""
```

```
print("=" * 60)
print("ERA5 ANALYSIS FOR KARIBA WIND STUDY")
print("=" * 60)
# Initialize downloader
downloader = ERA5Downloader()
# Ask user what to do
print("\nWhat would you like to do?")
print("1. Download ERA5 data (requires CDS API key)")
print("2. Process existing ERA5 files")
print("3. Quick test (no download)")
choice = input("\nEnter choice (1-3): ").strip()
if choice == '1':
# Download data
start_year = input(f"Start year [{downloader.config['project']['start_year']}]: ").strip()
end_year = input(f"End year [{downloader.config['project']['end_year']}]: ").strip()
start_year = int(start_year) if start_year else downloader.config['project']['start_year']
end_year = int(end_year) if end_year else downloader.config['project']['end_year']
print(f"\nDownloading ERA5 data for {start_year}-{end_year}...")
print("This may take a while (especially for 30 years).")
print("Make sure you have configured ~/.cdsapirc with your API key.")
confirm = input("\nContinue? (y/n): ").strip().lower()
if confirm != 'y':
print("Download cancelled.")
return
# Download data
files = downloader.download_data(start_year, end_year)
if files:
# Process downloaded data
ds = downloader.process_downloaded_data(files)
if ds is not None:
# Extract Charara data
df = downloader.extract_charara_data(ds)
# Create plots
downloader.create_basic_plots(df)
print("\n" + "=" * 60)
print("ANALYSIS COMPLETE!")
print("=" * 60)
print(f"Outputs saved in: {downloader.project_dir / 'outputs'}")
elif choice == '2':
# Process existing files
print("\nProcessing existing ERA5 files...")
ds = downloader.process_downloaded_data()
if ds is not None:
df = downloader.extract_charara_data(ds)
downloader.create_basic_plots(df)
print("\n" + "=" * 60)
print("PROCESSING COMPLETE!")
print("=" * 60)
elif choice == '3':
# Quick test
```

```
print("\nRunning quick test...")
print(f"Project directory: {downloader.project_dir}")
print(f"Python: {sys.executable}")
# Test imports
test_packages = ['numpy', 'pandas', 'xarray', 'matplotlib', 'cdsapi']
for package in test_packages:
try:
__import__(package)
print(f"✓ {package}")
except ImportError:
print(f"✗ {package}")
print("\nTest complete!")
else:
print("Invalid choice.")

if __name__ == "__main__":
main()
#EOF

#2echo "✓ Created corrected scripts/era5_analysis_01.py"

# Now run it
#python scripts/era5_analysis_01.py
```