Data Sources:

https://fred.stlouisfed.org/series/UNRATENSA https://fred.stlouisfed.org/series/CES5000000003 https://fred.stlouisfed.org/series/CEU5000000001 https://fred.stlouisfed.org/series/JTU5100JOL

#########################################################################

# Working Directory Setup & Imports

#########################################################################

#########################################################################

# Working Directory Setup & Imports

#########################################################################

import os

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import statsmodels.api as sm

from statsmodels.tsa.holtwinters import ExponentialSmoothing

from statsmodels.tsa.arima.model import ARIMA

from sklearn.model\_selection import train\_test\_split, TimeSeriesSplit

from statsmodels.stats.outliers\_influence import variance\_inflation\_factor

from sktime.performance\_metrics.forecasting import mean\_absolute\_percentage\_error

from sktime.forecasting.naive import NaiveForecaster

from sktime.forecasting.base import ForecastingHorizon

from statsmodels.tools.eval\_measures import rmse, meanabs, mse

from sklearn.metrics import r2\_score

from datetime import datetime

# Set working directory

os.chdir(r'/Users/apple/Desktop/University of Dallas/BANA 7350/final project/final\_data')

# Load data

df = pd.read\_csv('final\_data.csv', index\_col=0, parse\_dates=True)

df.index.freq = 'MS' # Ensure the index frequency is set to Monthly Start

df.dropna(inplace=True)

# General plot settings

plt.rcParams['lines.linewidth'] = 3

plt.rcParams['figure.figsize'] = [14.0, 5.0]

plt.rcParams['font.size'] = 18

plt.style.use('fivethirtyeight')

#########################################################################

# Exploratory Data Analysis and Correlations

#########################################################################

import matplotlib.pyplot as plt

from statsmodels.tsa.ar\_model import AutoReg, ar\_select\_order

# Visualize pairplot

sns.pairplot(df, x\_vars=['Average Hourly Earnings', 'Job Openings', 'Unemployment Rate'], y\_vars='Employees', height=4)

plt.suptitle("Relationship Between Economic Indicators and Tech Employment", y=1.02) # Adjust y for spacing

plt.show()

# Correlation heatmap

sns.heatmap(df.corr(), cmap="YlGnBu", annot=True)

plt.title('Correlation Matrix')

plt.show()

def tsplot(y, lags=None, style='fivethirtyeight'):

import pandas as pd

import numpy as np

import statsmodels.tsa.api as smt

import statsmodels.api as sm

import scipy.stats as scs

import matplotlib.pyplot as plt

if not isinstance(y, pd.Series):

y = pd.Series(y)

with plt.style.context(style):

fig = plt.figure(figsize=(10, 8)) # Set figsize directly in the plt.figure() call

layout = (3, 2)

ts\_ax = plt.subplot2grid(layout, (0, 0), colspan=2)

acf\_ax = plt.subplot2grid(layout, (1, 0))

pacf\_ax = plt.subplot2grid(layout, (1, 1))

qq\_ax = plt.subplot2grid(layout, (2, 0))

pp\_ax = plt.subplot2grid(layout, (2, 1))

y.plot(ax=ts\_ax)

ts\_ax.set\_title('Time Series Analysis Plots')

smt.graphics.plot\_acf(y, lags=lags, ax=acf\_ax, alpha=0.05)

smt.graphics.plot\_pacf(y, lags=lags, ax=pacf\_ax, alpha=0.05)

sm.qqplot(y, line='s', ax=qq\_ax)

qq\_ax.set\_title('QQ Plot')

scs.probplot(y, sparams=(y.mean(), y.std()), plot=pp\_ax)

plt.tight\_layout()

return

# Diagnostic plot for Employees data

tsplot(df['Employees'])

import statsmodels.api as sm

from statsmodels.graphics.tsaplots import plot\_acf, plot\_pacf

from datetime import datetime, timedelta

from statsmodels.tsa.seasonal import seasonal\_decompose

# Load the data

df = pd.read\_csv('final\_data.csv', index\_col=0, parse\_dates=True)

df.index.freq = 'MS' # Fixing the index frequency to Monthly

# Plot the data to observe trends

df['Employees'].plot(title='Monthly Employees Data')

# Decompose to find the Trend component

decomp = seasonal\_decompose(df['Employees'], model='additive')

trend = decomp.trend

# Calculate and plot 12-month rolling trend

trend\_12mo = trend.rolling(12).mean()

df['Employees'].plot(label='Employees')

trend\_12mo.plot(label='12-Month Rolling Trend')

plt.legend()

plt.title("Monthly Employees Data with 12-Month Rolling Trend")

plt.xlabel("Date") # Label for x-axis

plt.ylabel("Employees (Thousands)") # Label for y-axis

plt.show()

#########################################################################

# Linear Regression Model with Statsmodels

#########################################################################

# Define target and predictors

X = df[['Average Hourly Earnings', 'Job Openings', 'Unemployment Rate']]

y = df['Employees']

# Split data

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, train\_size=0.7, random\_state=100, shuffle=True)

# Add constant to training data

X\_train\_sm = sm.add\_constant(X\_train)

model = sm.OLS(y\_train, X\_train\_sm)

fit = model.fit()

# Summary of the model

print(fit.summary())

# Calculate Variance Inflation Factor (VIF)

vif = pd.DataFrame()

vif["Variable"] = X\_train.columns

vif["VIF"] = [variance\_inflation\_factor(X\_train.values, i) for i in range(X\_train.shape[1])]

print(vif)

#########################################################################

# Residual Analysis and Model Evaluation

#########################################################################

# Residual plot

residuals = y\_train - fit.fittedvalues

plt.scatter(fit.fittedvalues, residuals)

plt.axhline(0, color='red', linestyle='--')

plt.title('Residual Plot')

plt.show()

# Histogram of residuals

sns.histplot(residuals, kde=True)

plt.title('Histogram of Residuals')

plt.show()

# Predict on test data

X\_test\_sm = sm.add\_constant(X\_test)

y\_test\_pred = fit.predict(X\_test\_sm)

# Accuracy function

def accuracy(y\_true, y\_pred):

R\_sq = r2\_score(y\_true, y\_pred)

RMSE = rmse(y\_true, y\_pred)

MAE = meanabs(y\_true, y\_pred)

MSE = mse(y\_true, y\_pred)

MAPE = mean\_absolute\_percentage\_error(y\_true, y\_pred)

print(f'R-SQ: {R\_sq:.4f}, RMSE: {RMSE:.4f}, MAE: {MAE:.4f}, MSE: {MSE:.4f}, MAPE: {MAPE:.4f}')

accuracy(y\_test, y\_test\_pred)

#########################################################################

# Time Series Forecasting with SARIMAX

#########################################################################

import os

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from statsmodels.tsa.statespace.sarimax import SARIMAX

from statsmodels.tools.eval\_measures import rmse, mse

from sktime.performance\_metrics.forecasting import mean\_absolute\_percentage\_error

from sklearn.metrics import r2\_score, mean\_absolute\_error

# Fit SARIMAX model with exogenous variables

model = SARIMAX(y, exog=X, order=(2, 1, 5), seasonal\_order=(0, 1, 1, 12))

fit = model.fit()

print(fit.summary())

# Define forecast period and create index for 24-month forecast

n\_periods = 24

date\_range = pd.date\_range(start=X.index[-1] + pd.DateOffset(months=1), periods=n\_periods, freq='MS')

# Create exogenous variables for the forecast period using the last 12-month averages

X\_fcast\_values = df[['Average Hourly Earnings', 'Job Openings', 'Unemployment Rate']].tail(12).mean()

X\_fcast = pd.DataFrame([X\_fcast\_values] \* n\_periods, index=date\_range, columns=X.columns)

# Make the prediction with the exogenous forecast values

y\_fcast = fit.predict(start=len(y), end=len(y) + n\_periods - 1, exog=X\_fcast)

# Plot the forecast along with historical data

plt.figure(figsize=(14, 7))

y\_train.plot(label='Train')

y\_test.plot(label='Test')

y\_fcast[:len(y\_test)].plot(label='SARIMAX Forecast', color='orange') # Forecast for test horizon

plt.legend()

plt.title('SARIMAX Forecast')

plt.xlabel('Date')

plt.ylabel('Employees (Thousands of Persons)')

plt.show()

# Calculate accuracy metrics for the forecast period

# Assuming the last 24 months of actual data as `y\_test`

y\_test = y[-24:] # Select the last 24 months of actual data for comparison

# Define the accuracy function

def accuracy(y\_true, y\_pred):

R\_sq = r2\_score(y\_true, y\_pred)

RMSE = rmse(y\_true, y\_pred)

MAE = mean\_absolute\_error(y\_true, y\_pred)

MSE = mse(y\_true, y\_pred)

MAPE = mean\_absolute\_percentage\_error(y\_true, y\_pred)

print(f'R-SQ : {R\_sq:.4f}, RMSE : {RMSE:.4f}, MAE : {MAE:.4f}, MSE : {MSE:.4f}, MAPE : {MAPE:.4f}')

# Run accuracy metrics on the forecast vs actuals

accuracy(y\_test, y\_fcast[:len(y\_test)])

# Obtain the forecast data with confidence intervals for the 24-month forecast period

forecast\_frame = y\_fcast.to\_frame(name='Predicted')

forecast\_frame['Lower CI'] = forecast\_frame['Predicted'] - 1.96 \* fit.bse # Lower confidence interval

forecast\_frame['Upper CI'] = forecast\_frame['Predicted'] + 1.96 \* fit.bse # Upper confidence interval

# Set up the index for the forecast period

forecast\_frame.index = date\_range

forecast\_frame.index.name = 'Month'

# Combine with the historical data to have a full dataset

full\_data\_with\_forecast = pd.concat([df, forecast\_frame], axis=1)

full\_data\_with\_forecast.to\_csv('forecast\_output\_2026.csv', index=True) # Ensure the extension is '.csv'

#########################################################################

# Forecasting with Holt-Winters' Seasonal Smoothing

#########################################################################

from statsmodels.tsa.holtwinters import ExponentialSmoothing

# Split data into training and testing sets

train\_size = int(len(df) \* 0.7)

train = df['Employees'][:train\_size]

test = df['Employees'][train\_size:]

# Holt-Winters seasonal model (fitted on training data)

model\_hw = ExponentialSmoothing(train, seasonal='add', seasonal\_periods=12, initialization\_method="estimated").fit()

forecast\_hw = model\_hw.forecast(len(test)) # Forecast for the length of the test set

# Extend the forecast beyond the test set by an additional 24 months if needed

future\_forecast\_hw = model\_hw.forecast(len(test) + 24) # Forecast for test set + 24 months

# Plot Actual, Train, Test, and Forecasted data

plt.figure(figsize=(14, 7))

train.plot(label='Train')

test.plot(label='Test')

future\_forecast\_hw.plot(label='Holt-Winters Forecast ', color='orange')

plt.legend()

plt.title('24 Month Holt-Winters Forecast')

plt.xlabel('Date')

plt.ylabel('Employees (Thousands of Persons)')

plt.show()

def accuracy(y\_true, y\_pred):

# Calculate the accuracy metrics

R\_sq = r2\_score(y\_true, y\_pred)

RMSE = rmse(y\_true, y\_pred)

MAE = meanabs(y\_true, y\_pred)

MSE = mse(y\_true, y\_pred)

MAPE = mean\_absolute\_percentage\_error(y\_true, y\_pred)

# Print the results in a formatted way

print(f'R-SQ : {R\_sq:.4f}, RMSE : {RMSE:.4f}, MAE : {MAE:.4f}, MSE : {MSE:.4f}, MAPE : {MAPE:.4f}')

accuracy(test, forecast\_hw)

#########################################################################

# Seasonal Naive Forecasting (for baseline comparison)

#########################################################################

from sktime.forecasting.naive import NaiveForecaster

from sktime.forecasting.base import ForecastingHorizon

import pandas as pd

import matplotlib.pyplot as plt

from sktime.forecasting.naive import NaiveForecaster

from sktime.forecasting.base import ForecastingHorizon

from sktime.utils.plotting import plot\_series, plot\_correlations

# Load your data

df = pd.read\_csv('final\_data.csv', index\_col=0, parse\_dates=True)

y = df['Employees']

# Convert index to PeriodIndex with monthly frequency

y.index = pd.PeriodIndex(y.index, freq="M")

train\_size = int(len(y) \* 0.70)

y\_train = y[0:train\_size]

y\_test = y[train\_size:]

y\_train

y\_test

# now lets plot the train and test data:

plot\_series(y\_train, y\_test, labels=["Train", "Test"])

plot\_correlations(y, lags=24, alpha=0.05)

# set up test horizon:

# Initially, our forecast horizon will lie within the same time frame as y\_test

# So, set the relative dates option off, i.e., is\_relative=false, since you will be using exact dates from y\_test index

test\_horizon = ForecastingHorizon(y\_test.index, is\_relative=False)

test\_horizon

# doing the same for the train horizon i.e., the date range within the train data set

train\_horizon = ForecastingHorizon(y\_train.index, is\_relative=False)

train\_horizon

# Set up forecasting horizon for the next 24 months from the end of the actual data

future\_horizon = pd.period\_range(start=y.index[-1] + 1, periods=24, freq="M")

future\_horizon = ForecastingHorizon(future\_horizon, is\_relative=False)

# Create the Seasonal Naive model with monthly seasonality

model\_snaive = NaiveForecaster(strategy='last', sp=12) # sp=12 for monthly seasonality

fit\_snaive = model\_snaive.fit(y) # Fitting on the entire dataset

fcast\_snaive = fit\_snaive.predict(future\_horizon)

# Plot the actual data and forecast

plt.figure(figsize=(14, 7))

y\_train.plot(label='Train')

y\_test.plot(label='Test')

fcast\_snaive.plot(label='Seasonal Naive Forecast ', color='orange')

plt.legend()

plt.title('24 Month Seasonal Naive Forecast ')

plt.xlabel('Date')

plt.ylabel('Employees (Thousands of Persons)')

plt.show()

y\_test\_adjusted = y\_test[:len(fcast\_snaive)]

# Obtain fitted values forSeasonal Naive within the training horizon

y\_fitted\_snaive = fit\_snaive.predict(train\_horizon)

# Remove NaN values using forward fill (or backward fill) instead of filling with 0

y\_fitted\_snaive.fillna(method='ffill', inplace=True)

def accuracy(y\_test,y\_pred):

from statsmodels.tools.eval\_measures import rmse, rmspe, meanabs, mse

from sktime.performance\_metrics.forecasting import mean\_absolute\_percentage\_error

R\_sq = r2\_score(y\_test, y\_pred)

RMSE = rmse(y\_test,y\_pred)

RMSPE = rmspe(y\_test,y\_pred)

MAE = meanabs(y\_test,y\_pred)

MSE = mse(y\_test,y\_pred)

MAPE = mean\_absolute\_percentage\_error(y\_test,y\_pred)

print(f'R-SQ : {R\_sq:.4f}, RMSE : {RMSE:.4f}, MAE : {MAE:.4f}, MSE : {MSE:.4f}, MAPE : {MAPE:.4f}')

y\_test\_adjusted = y\_test[:len(fcast\_snaive)]

accuracy(y\_test\_adjusted, fcast\_snaive)

y\_fitted\_snaive = fit\_snaive.predict(train\_horizon)

# Removing all the NaN values in the fitted data series

y\_fitted\_snaive.fillna(0,inplace=True)

# now use the Accuracy function to measure the accuracy of all the model fits we have created:

accuracy(y\_train,y\_fitted\_snaive)

A screenshot of a computer screen

Description automatically generated

A graph with blue dots

Description automatically generatedA blue squares with white text

Description automatically generatedA screenshot of a graph

Description automatically generatedA graph with blue dots and red line

Description automatically generatedA graph with a curve

Description automatically generatedA graph of a graph showing the number of the same graph

Description automatically generated with medium confidenceA graph showing the growth of the stock market

Description automatically generatedA graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of

Description automatically generatedA graph of a graph showing the fall of the stock market

Description automatically generated