Project2_TimeSeries_Task2

December 27, 2024

0.0.1 Task 2

[1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
from statsmodels.tsa.seasonal import seasonal_decompose
import plotly.graph_objects as go
from statsmodels.tsa.holtwinters import ExponentialSmoothing
from sklearn.metrics import mean_absolute_error, mean_squared_error
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.arima.model import ARIMA
import itertools
import statsmodels.api as sm

import warnings
warnings.filterwarnings('ignore')
```

[2]:
```
data_2020 = pd.read_csv(r"/Users/shubhamgaur/Desktop/NU/Foundation Data␣
 ↪Analytics/Project 2 Time Series/2020_US_Region_Mobility_Report.csv")
data_2021 = pd.read_csv(r"/Users/shubhamgaur/Desktop/NU/Foundation Data␣
 ↪Analytics/Project 2 Time Series/2021_US_Region_Mobility_Report.csv")
data_2022 = pd.read_csv(r"/Users/shubhamgaur/Desktop/NU/Foundation Data␣
 ↪Analytics/Project 2 Time Series/2022_US_Region_Mobility_Report.csv")
```

Data Filtering

[3]:
```
data_2020 = data_2020.
 ↪drop(columns=['retail_and_recreation_percent_change_from_baseline',␣
 ↪'parks_percent_change_from_baseline',␣
 ↪'transit_stations_percent_change_from_baseline', 'metro_area',␣
 ↪'iso_3166_2_code'])
data_2021 = data_2021.
 ↪drop(columns=['retail_and_recreation_percent_change_from_baseline',␣
 ↪'parks_percent_change_from_baseline',␣
 ↪'transit_stations_percent_change_from_baseline', 'metro_area',␣
 ↪'iso_3166_2_code'])
```

```
data_2022 = data_2022.
↪drop(columns=['retail_and_recreation_percent_change_from_baseline',␣
↪'parks_percent_change_from_baseline',␣
↪'transit_stations_percent_change_from_baseline', 'metro_area',␣
↪'iso_3166_2_code'])
```

[4]:
```
king_county_2020 = data_2020[data_2020['sub_region_2'] == 'King County']
king_county_2021 = data_2021[data_2021['sub_region_2'] == 'King County']
king_county_2022 = data_2022[data_2022['sub_region_2'] == 'King County']
```

Task 2.1: Put together your entire time series using all the data from 2020-2022. You should end up with 1 dataframe that contains all the data points.

[5]:
```
df = pd.concat([king_county_2020, king_county_2021, king_county_2022],␣
↪ignore_index=True)
```

[6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 974 entries, 0 to 973
Data columns (total 10 columns):
 #   Column                                            Non-Null Count  Dtype
---  ------                                            --------------  -----
 0   country_region_code                               974 non-null    object
 1   country_region                                    974 non-null    object
 2   sub_region_1                                      974 non-null    object
 3   sub_region_2                                      974 non-null    object
 4   census_fips_code                                  974 non-null    float64
 5   place_id                                          974 non-null    object
 6   date                                              974 non-null    object
 7   grocery_and_pharmacy_percent_change_from_baseline 974 non-null    float64
 8   workplaces_percent_change_from_baseline           974 non-null    float64
 9   residential_percent_change_from_baseline          974 non-null    float64
dtypes: float64(4), object(6)
memory usage: 76.2+ KB
```

Task 2.2: Trim down your time series to remove the months before April 2020. This will remove the very early pandemic and the pre-pandemic conditions.

[7]:
```
df['date'] = pd.to_datetime(df['date'])
df = df[df['date'] >= '2020-04-01']
df = df.reset_index(drop=True)
```

Task 2.3: For each of the 3 time series, perform an additive Time Series Decomposition and plot the results. You should show the trend, seasonality, and remainder in your plots.

[8]:
```
df.set_index('date', inplace=True)

# Perform additive time series decomposition for each column
```

```python
for column in df.columns[6:]:  # Assuming the relevant columns start from index␣
↪7
    result = seasonal_decompose(df[column], model='additive')

    # Create Plotly figure for decomposition
    fig = go.Figure()

    # Add trend component
    fig.add_trace(go.Scatter(x=result.trend.index, y=result.trend,␣
↪mode='lines', name='Trend'))

    # Add seasonality component
    fig.add_trace(go.Scatter(x=result.seasonal.index, y=result.seasonal,␣
↪mode='lines', name='Seasonality'))

    # Add remainder component
    fig.add_trace(go.Scatter(x=result.resid.index, y=result.resid,␣
↪mode='lines', name='Remainder'))

    # Update layout
    fig.update_layout(title=f'Time Series Decomposition - {column}',␣
↪xaxis_title='Date', yaxis_title='Value')
    fig.show()
```

Task 2.4: For each time series, build a forecasting model using Exponential Smoothing (ES). You should test out at least 2 different ES models and use forecast evaluation metrics (e.g. MAE, RMSE) to demonstrate why you chose your best ES model

[9]:
```python
# List of time series columns
time_series_columns = ['grocery_and_pharmacy_percent_change_from_baseline',␣
↪'workplaces_percent_change_from_baseline',␣
↪'residential_percent_change_from_baseline']

# Train-test split (you can adjust the split point as needed)
train_size = int(len(df) * 0.8)

for column in time_series_columns:
    # Select the time series column
    y = df[column]

    # Train-test split
    train, test = y[:train_size], y[train_size:]

    # Model 1: Simple Exponential Smoothing (SES)
    model_ses = ExponentialSmoothing(train, trend='add', seasonal=None)
    fit_ses = model_ses.fit()
    forecast_ses = fit_ses.forecast(len(test))
```

```python
    # Model 2: Holt-Winters Exponential Smoothing (Holt-Winters)
    model_hw = ExponentialSmoothing(train, trend='add', seasonal='add', seasonal_periods=12)
    fit_hw = model_hw.fit()
    forecast_hw = fit_hw.forecast(len(test))

    # Evaluate models
    mae_ses = mean_absolute_error(test, forecast_ses)
    rmse_ses = np.sqrt(mean_squared_error(test, forecast_ses))

    mae_hw = mean_absolute_error(test, forecast_hw)
    rmse_hw = np.sqrt(mean_squared_error(test, forecast_hw))

    # Print evaluation metrics
    print(f"\nTime Series - {column}")
    print(f"SES MAE: {mae_ses:.2f}, RMSE: {rmse_ses:.2f}")
    print(f"Holt-Winters MAE: {mae_hw:.2f}, RMSE: {rmse_hw:.2f}")

    # Plot the results using Plotly
    fig = go.Figure()

    # Training Data
    fig.add_trace(go.Scatter(x=train.index, y=train, mode='lines', name='Training Data'))

    # Test Data
    fig.add_trace(go.Scatter(x=test.index, y=test, mode='lines', name='Test Data'))

    # SES Forecast
    fig.add_trace(go.Scatter(x=test.index, y=forecast_ses, mode='lines', name='SES Forecast'))

    # Holt-Winters Forecast
    fig.add_trace(go.Scatter(x=test.index, y=forecast_hw, mode='lines', name='Holt-Winters Forecast'))

    # Update layout
    fig.update_layout(title=f'Time Series Forecasting - {column}', xaxis_title='Date', yaxis_title='Value')
    fig.show()

    # Choose the best model based on evaluation metrics
    best_model = 'SES' if mae_ses < mae_hw else 'Holt-Winters'
    print(f"The best model for {column} is: {best_model}\n")
```

```
Time Series - grocery_and_pharmacy_percent_change_from_baseline
SES MAE: 5.37, RMSE: 6.03
Holt-Winters MAE: 5.50, RMSE: 6.20

The best model for grocery_and_pharmacy_percent_change_from_baseline is: SES


Time Series - workplaces_percent_change_from_baseline
SES MAE: 16.44, RMSE: 17.70
Holt-Winters MAE: 17.10, RMSE: 18.74

The best model for workplaces_percent_change_from_baseline is: SES


Time Series - residential_percent_change_from_baseline
SES MAE: 9.14, RMSE: 10.60
Holt-Winters MAE: 9.16, RMSE: 10.63

The best model for residential_percent_change_from_baseline is: SES
```

Task 2.5: For each time series, build a forecasting model using ARIMA. You must show why you chose your ARIMA model.

[10]:
```python
best_orders = {}
for column in time_series_columns:
    print(f"Finding best ARIMA order for {column}")

    y = df[column]

    best_aic = np.inf
    best_order = None

    p = d = q = range(0, 5)
    pdq = list(itertools.product(p, d, q))

    for param in pdq:
        try:
            model_arima = sm.tsa.ARIMA(y, order=param)
            model_arima_fit = model_arima.fit()
            aic = model_arima_fit.aic
            if aic < best_aic:
                best_aic = aic
                best_order = param
        except Exception as e:
            print(f"Failed for {param}: {e}")
            continue

    best_orders[column] = best_order
```

```
    print(f"Best ARIMA Model Order for {column}: {best_order} (AIC:␣
↪{best_aic})")

# Print Best Model Orders
print("\nBest ARIMA Model Orders:")
for column, order in best_orders.items():
    print(f"{column}: {order}")
```

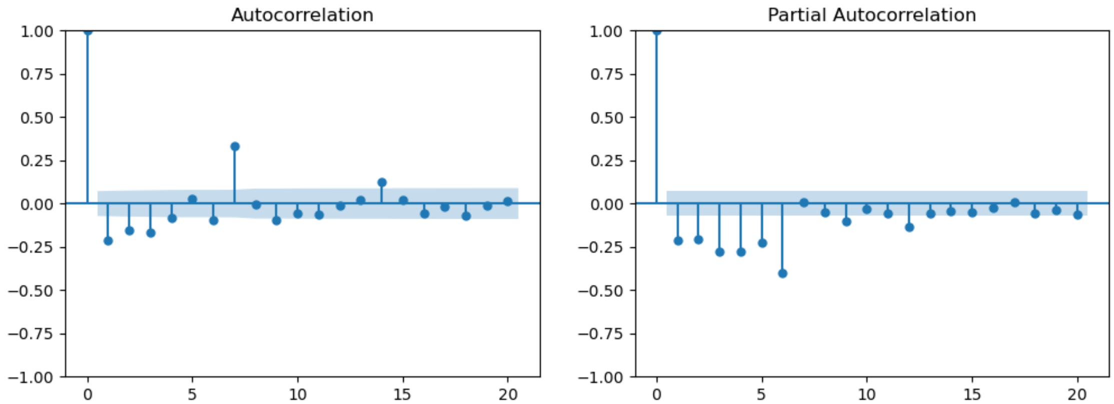
```
Finding best ARIMA order for grocery_and_pharmacy_percent_change_from_baseline
Best ARIMA Model Order for grocery_and_pharmacy_percent_change_from_baseline:
(4, 3, 2) (AIC: 14.0)
Finding best ARIMA order for workplaces_percent_change_from_baseline
Best ARIMA Model Order for workplaces_percent_change_from_baseline: (4, 0, 4)
(AIC: 6678.173091037616)
Finding best ARIMA order for residential_percent_change_from_baseline
Best ARIMA Model Order for residential_percent_change_from_baseline: (4, 1, 4)
(AIC: 4125.166001301649)

Best ARIMA Model Orders:
grocery_and_pharmacy_percent_change_from_baseline: (4, 3, 2)
workplaces_percent_change_from_baseline: (4, 0, 4)
residential_percent_change_from_baseline: (4, 1, 4)
```

[11]:

```
# Best ARIMA orders
arima_orders = {
    'grocery_and_pharmacy_percent_change_from_baseline': (4, 3, 2),
    'workplaces_percent_change_from_baseline': (4, 0, 4),
    'residential_percent_change_from_baseline': (4, 1, 4)
}

for column in time_series_columns:
    # Select the time series column
    y = df[column]

    # Train-test split
    train, test = y[:train_size], y[train_size:]

    # Check stationarity using Augmented Dickey-Fuller test
    result_adf = adfuller(train)
    print(f'\n\nADF Statistic: {result_adf[0]}, p-value: {result_adf[1]}')
    if result_adf[1] > 0.05:
        print('The series is not stationary. Applying differencing...')
        train_diff = train.diff().dropna()
    else:
        print('The series is stationary.')
        train_diff = train
```

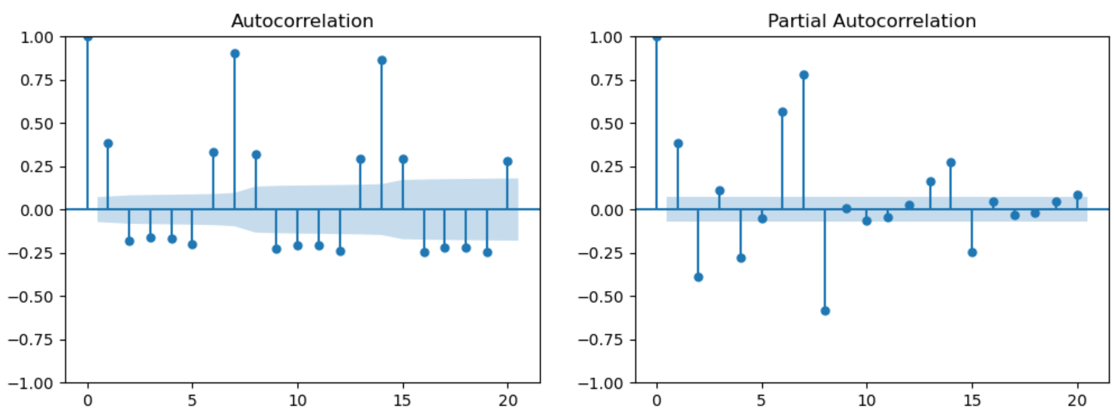
```
    # Plot ACF and PACF to determine ARIMA parameters
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))
    plot_acf(train_diff, lags=20, ax=ax1)
    plot_pacf(train_diff, lags=20, ax=ax2)
    plt.show()

    # Choose p, d, and q based on ACF and PACF plots
    p, d, q = arima_orders[column]

    # Build ARIMA model
    model = ARIMA(train, order=(p, d, q))
    fit_arima = model.fit()

    # Forecast using ARIMA model
    forecast_arima = fit_arima.predict(start=len(train), end=len(train) +␣
↪len(test) - 1, typ='levels')

    # Evaluate the model
    mae_arima = mean_absolute_error(test, forecast_arima)
    rmse_arima = np.sqrt(mean_squared_error(test, forecast_arima))

    # Print evaluation metrics
    print(f"\nTime Series - {column}")
    print(f"ARIMA MAE: {mae_arima:.2f}, RMSE: {rmse_arima:.2f}")

    # Plot the results using Plotly
    fig = go.Figure()

    # Training Data
    fig.add_trace(go.Scatter(x=train.index, y=train, mode='lines',␣
↪name='Training Data'))

    # Test Data
    fig.add_trace(go.Scatter(x=test.index, y=test, mode='lines', name='Test␣
↪Data'))

    # ARIMA Forecast
    fig.add_trace(go.Scatter(x=test.index, y=forecast_arima, mode='lines',␣
↪name='ARIMA Forecast'))

    # Update layout
    fig.update_layout(title=f'Time Series Forecasting - {column}',␣
↪xaxis_title='Date', yaxis_title='Value')
    fig.show()
```

```
ADF Statistic: -2.609969597393196, p-value: 0.09095421120458902
The series is not stationary. Applying differencing…
```



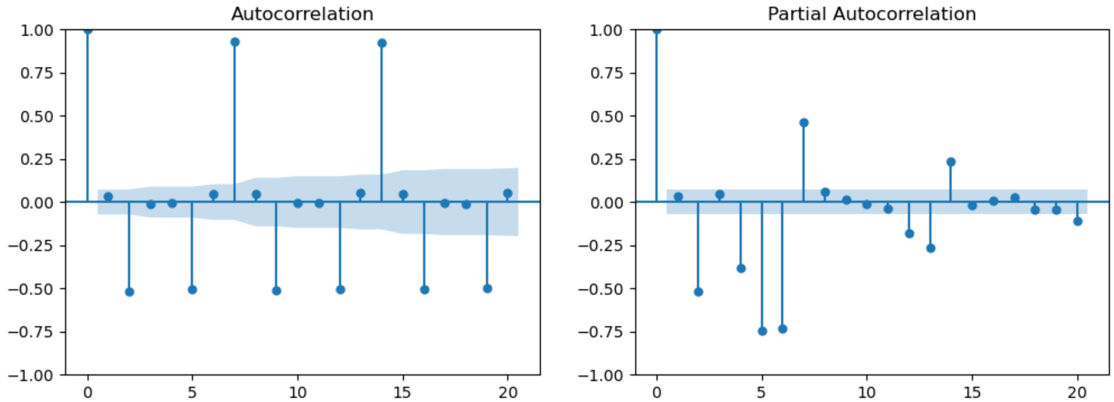

```
Time Series - grocery_and_pharmacy_percent_change_from_baseline
ARIMA MAE: 9.35, RMSE: 10.14
```

```
ADF Statistic: -3.260853797096699, p-value: 0.016721602304220127
The series is stationary.
```




```
Time Series - workplaces_percent_change_from_baseline
ARIMA MAE: 8.81, RMSE: 10.80
```

```
ADF Statistic: -2.4518282416425845, p-value: 0.12763912938564714
The series is not stationary. Applying differencing…
```




```
Time Series - residential_percent_change_from_baseline
ARIMA MAE: 1.89, RMSE: 2.54
```

Task 2.6: Compare your best ES and best ARIMA models for each time series using forecast evaluation metrics. Show which model is best in each case. For grocery_and_pharmacy_percent_change_from_baseline time series:

SES MAE: 5.37, RMSE: 6.03 Holt-Winters MAE: 5.50, RMSE: 6.20 ARIMA MAE: 9.35, RMSE: 10.14 In this case, both SES and Holt-Winters outperform ARIMA based on lower MAE and RMSE values.

For workplaces_percent_change_from_baseline time series:

SES MAE: 16.44, RMSE: 17.70 Holt-Winters MAE: 17.10, RMSE: 18.74 ARIMA MAE: 8.81, RMSE: 10.80 ARIMA outperforms both SES and Holt-Winters with significantly lower MAE and RMSE values.

For residential_percent_change_from_baseline time series:

SES MAE: 9.14, RMSE: 10.60 Holt-Winters MAE: 9.16, RMSE: 10.63 ARIMA MAE: 1.89, RMSE: 2.54 ARIMA again outperforms both SES and Holt-Winters with significantly lower MAE and RMSE values.

Conclusion:

For grocery_and_pharmacy_percent_change_from_baseline, SES or Holt-Winters may be preferred over ARIMA. For workplaces_percent_change_from_baseline and residential_percent_change_from_baseline, ARIMA appears to be the better choice based on lower MAE and RMSE.

Task 2.7: Using your best model, forecast the rest of 2022 for each time series. Show these forecasts by plotting the past data points in 1 color and the future data points in a second color

[12]:
```
df.reset_index(inplace=True)
```

```
# ARIMA orders and best model information from Task 2.5
arima_orders = {
    'workplaces_percent_change_from_baseline': (4, 0, 4),
    'residential_percent_change_from_baseline': (4, 1, 4)
}

best_models = {
    'grocery_and_pharmacy_percent_change_from_baseline': 'SES',  # or␣
↪'Holt-Winters'
    'workplaces_percent_change_from_baseline': 'ARIMA',
    'residential_percent_change_from_baseline': 'ARIMA'
}

# Assuming your dataframe is named df and has the necessary columns including␣
↪'date'
# Also, make sure the 'date' column is in datetime format
df['date'] = pd.to_datetime(df['date'])

# Loop through each time series column
for column in best_models.keys():
    # Extract the time series data
    y = df[['date', column]].set_index('date')

    # Forecast the remaining data for 2022 based on the best model
    if best_models[column] == 'ARIMA':
        best_order = arima_orders[column]
        model_arima = sm.tsa.ARIMA(y, order=best_order)
        model_arima_fit = model_arima.fit()
        forecast_index = pd.date_range(start=y.index.max() + pd.
↪Timedelta(days=1), end='2022-12-31', freq='D')
        forecast = model_arima_fit.forecast(steps=len(forecast_index))
    else:
        # Assuming SES/Holt-Winters
        model = ExponentialSmoothing(y, trend='add', seasonal='add',␣
↪seasonal_periods=12)
        fit = model.fit()
        forecast_index = pd.date_range(start=y.index.max() + pd.
↪Timedelta(days=1), end='2022-12-31', freq='D')
        forecast = fit.forecast(steps=len(forecast_index))

    # Plot past and future data points using Plotly
    fig = go.Figure()

    # Past Data
    fig.add_trace(go.Scatter(x=y.index, y=y[column], mode='lines', name='Past␣
↪Data', line=dict(color='blue')))
```

```
    # Forecast
    fig.add_trace(go.Scatter(x=forecast_index, y=forecast, mode='lines',␣
↪name='Forecast', line=dict(color='green')))

    # Update layout
    fig.update_layout(title=f'{column} Forecast for the Rest of 2022',␣
↪xaxis_title='Date', yaxis_title='Value')
    fig.show()
```

[ ]:

```

```