National College of Ireland

Air Passenger and Freight Demand Forecast for Ireland

MSc Research Project - Configuration Manual
Data Analytics

Nivriti Verma
Student ID: 21201421

School of Computing
National College of Ireland

Supervisor: Dr. Hicham Rifai

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Nivriti Verma
Student ID:	21201421
Programme:	Data Analytics
Year:	2023
Module:	MSc Research Project - Configuration Manual
Supervisor:	Dr. Hicham Rifai
Submission Due Date:	14/12/2023
Project Title:	Air Passenger and Freight Demand Forecast for Ireland
Word Count:	XXX
Page Count:	12

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Nivriti Verma
Date:	14th December 2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Air Passenger and Freight Demand Forecast for Ireland

Nivriti Verma
21201421

1 Introduction

The configuration manual document outlines the procedures undertaken during the coding phase of the project "Air Passenger and Freight Demand Forecast for Ireland". Both hardware and software configurations are specified, along with the intricacies of the programming process. The document also mentions the requisite procedures for code execution and highlights the deployment stages for ensuring the effectiveness of the code.

2 System Configuration

2.1 Hardware Specifications

Figure 1 displays the hardware specifications on which the research is conducted.

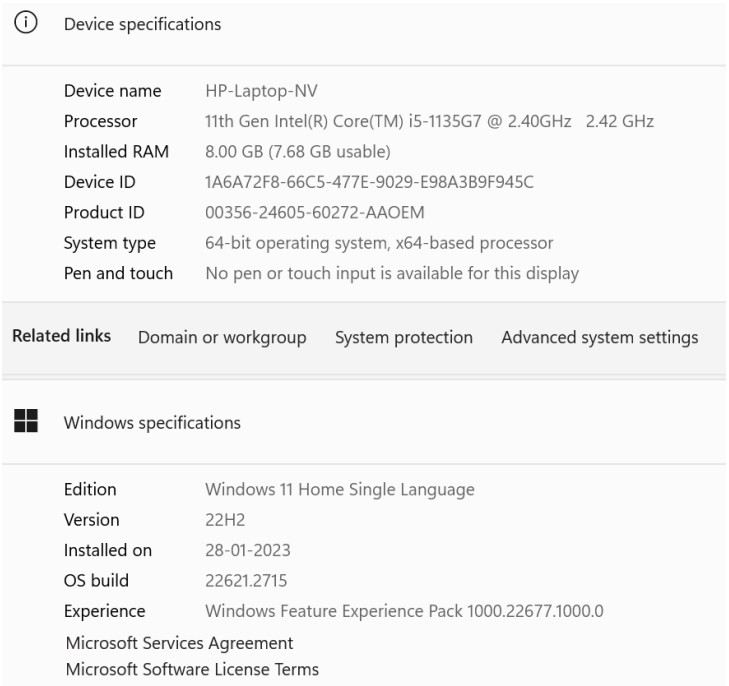

Figure 1: System Configuration

2.2 Software Specifications

This section summarises the software and its specifications.

2.2.1 Environment

Jupyter Notebook 6.5.4 version was employed as the primary coding environment for this research as shown in Figure 2. It offered collaborative and interactive features for conducting code deployment, data analysis and visualization.

```
C:\Users\Nivriti>jupyter notebook --version
6.5.4
```

Figure 2: Jupyter Notebook

2.2.2 Programming Language

Python version 3.10.9 was utilised to develop time series forecasting models for predicting passenger and freight demand forecast as shown in Figure 3. Due to the availability, of extensive libraries in Python and its readability, it was an ideal choice for programming language.

```
C:\Users\Nivriti>Python --version
Python 3.10.9
```

Figure 3: Python

3 Data Collection

The initial step is to load the Eurostat's downloaded dataset, in (csv) format. Passenger[1] and freight[2] dataframes are created to store the datasets. Figure 4 shows the code for loading the dataset.

Loading the Data

```
#Loading Freight and Mail Data
df_fnm = pd.read_csv("D:/DA_NCI/Sem_3/Research-Thesis/Datasets/Air transport measurement - passengers/df_fnm.csv")
df_fnm.head(5)
```

Figure 4: Loading dataset

[1]https://ec.europa.eu/eurostat/databrowser/view/avia_par_ie__custom_8975440/default/table?lang=en
[2]https://ec.europa.eu/eurostat/databrowser/view/avia_gor_ie__custom_8975476/default/table?lang=en

4 Data Preprocessing and Transformation

All the steps performed for data preprocessing and transformation are identical for passenger and freight datasets. This section exclusively presents the figures of the freight code along with an explanation to streamline the information and eliminate redundancy. Figure 5 shows the conversion of TIME_PERIOD column to datetime format.

```
#Convert time period column in df_ie to datetime format
df_ie['TIME_PERIOD'] = pd.to_datetime(df_ie['TIME_PERIOD'])
df_ie.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15431 entries, 0 to 15430
Data columns (total 9 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   DATAFLOW     15431 non-null  object
 1   LAST UPDATE  15431 non-null  object
 2   freq         15431 non-null  object
 3   unit         15431 non-null  object
 4   tra_meas     15431 non-null  object
 5   airp_pr      15431 non-null  object
 6   TIME_PERIOD  15431 non-null  datetime64[ns]
 7   OBS_VALUE    15431 non-null  int64
 8   OBS_FLAG     0 non-null      float64
dtypes: datetime64[ns](1), float64(1), int64(1), object(6)
memory usage: 1.1+ MB
```

Figure 5: Converting time period column to datetime format

Null values are checked thereafter in Figure 6

```
#Checking for null values in df_fnm
nulls = df_fnm.isnull().sum().to_frame().sort_values(by=0, ascending = False)
nulls.columns = ["Missing values"]
nulls[nulls['Missing values'] != 0]
```

Figure 6: Checking for null values

Figure 7 displays the graph for total freight traffic generation from 2020 to 2023.

In Figure 8 airports with the highest freight traffic are grouped based on time period and partner airport's values so that their share (in %) is calculated along with the mean number of freight and mail per year.

```
#Airline Freight and Mail Traffic Generation for all the Ireland and partner airports

FNM_yr = df_fnm.groupby(["TIME_PERIOD"])["OBS_VALUE"].sum()

fig, ax = plt.subplots(figsize=(15,7))

#Plotting the main FNM line
sns.lineplot(x=FNM_yr.index, y=FNM_yr.values, markers=True, ax=ax,zorder=0)

# Looking for maximum FNM for each year
FNM_yr_max = FNM_yr.groupby(FNM_yr.index.year).max()
FNM_yr_max_complete = FNM_yr[FNM_yr.isin(FNM_yr_max.values)].to_frame()

# Marking points of interest
plt.scatter(FNM_yr_max_complete.index, FNM_yr_max_complete.values, color = "red", zorder=2)

#Annotating each marker
for t,v in FNM_yr_max_complete.reset_index().values:
    ax.text(t,v+90,int(v))

# Looking for minimum FNM for each year
FNM_yr_min = FNM_yr.groupby(FNM_yr.index.year).min()
FNM_yr_min_complete = FNM_yr[FNM_yr.isin(FNM_yr_min.values)].to_frame()
plt.scatter(FNM_yr_min_complete.index, FNM_yr_min_complete.values, color = "green", zorder=2)
for t,v in FNM_yr_min_complete.reset_index().values:
    ax.text(t,v-150,int(v))

plt.title("Freight and Mail Air Traffic Generation for all Ireland and Partner Airports", size = 20)
plt.xlabel("Time Period", fontweight="bold", size = 12)
plt.ylabel("Freight and Mail (in tonnes)", fontweight="bold", size = 12)
plt.show()
```

Figure 7: Graph for total freight traffic generation across 3 years

```
#For airports with highest FNM traffic generation, mean number of FNM/airport each year and its share in the total traffic is ger

FNM_airport_yr = df_fnm.groupby(["TIME_PERIOD","airp_pr"])["OBS_VALUE"].sum().divide(1000)
FNM_airport_yr = FNM_airport_yr.reset_index()
pivot_01 = FNM_airport_yr.pivot_table(values="OBS_VALUE", index="airp_pr", columns="TIME_PERIOD", fill_value=0)
```

```
#Top 3 airports which generate Most FNM Traffic

avg_FNM = pivot_01.mean(axis=1)
TOP3_airports_fnm = avg_FNM.nlargest(3).to_frame().mul(1000).astype("int64")
TOP3_airports_fnm.columns = ["Mean no. of FnM per year"]
sum_of_all = TOP3_airports_fnm.loc[:,"Mean no. of FnM per year"].sum()
TOP3_airports_fnm.loc[:,"Share [%]"] = TOP3_airports_fnm.loc[:,"Mean no. of FnM per year"].div(sum_of_all).mul(100).round(1)
TOP3_airports_fnm
```

Figure 8: Top 3 freight and mail traffic generation airports

```
#3 airports which generate least FNM traffic

avg_FNM = pivot_01.mean(axis=1)
Least_airports_fnm = avg_FNM.nsmallest(3).to_frame().mul(1000).astype("int64")
Least_airports_fnm.columns = ["Mean no. of FnM per year"]
sum_of_all = Least_airports_fnm.loc[:,"Mean no. of FnM per year"].sum()
Least_airports_fnm.loc[:,"Share [%]"] = Least_airports_fnm.loc[:,"Mean no. of FnM per year"].div(sum_of_all).mul(100).round(1)
Least_airports_fnm
```

Figure 9: 3 least freight and mail traffic generation airports

Conversely, airport pairs which generate least freight traffic are calculated in Figure 9.

Figure 10 displays the generated graph for the top 3 airports with maximum freight and mail traffic.

```
#Share of top 3 FNM traffic airports in a graph

airp_fnm = pivot_01.mean(axis=1)
top_fnm = airp_fnm.nlargest(3).index

df_top3_fnm = df_fnm[df_fnm['airp_pr'].isin(top_fnm)]

top_fnm_grouped = df_top3_fnm.groupby(["TIME_PERIOD", "airp_pr"])["OBS_VALUE"].sum().reset_index()

pivot_02 = top_fnm_grouped.pivot_table(values="OBS_VALUE", index="TIME_PERIOD", columns="airp_pr", fill_value=0)
pivot_02["Total"] = pivot_02.sum(axis=1)

for col in pivot_02.columns[:-1]:
    pivot_02["Share of " + str(col)] = pivot_02[col] / pivot_02["Total"]

ratios = pivot_02.iloc[:, -3:]

ratios.plot(figsize=(15, 10), title="Share of top 5 partner airports to total freight and mail traffic generated")
plt.show()
```

Figure 10: Top 3 airport with highest shares

Next, in Figure 11 a new dataframe is created to store the value of the top 3 airports and plots the graph to proceed with time series decomposition. Figure 12 and Figure 13 displays the code for generating graphs of additive and multiplicative decomposition time series.

Decomposition of Time Series

```
#Freight and Mail generation for top 3 airports

df_FNM = df_fnm_top3_filtered.groupby("TIME_PERIOD")["OBS_VALUE"].sum().to_frame()
df_fnm = df_fnm_top3_filtered.groupby("TIME_PERIOD")["OBS_VALUE"].sum().to_frame()
fig, ax = plt.subplots(1,1,figsize=(15,5))
df_FNM.plot(ax=ax)
ax.set_xlabel("Time Period")
ax.set_ylabel("Freight and Mail")
plt.grid(True)
plt.show()
```

Figure 11: Dataframe creation to store top 3 values

The function created that is used to test the stationarity using Dickey-Fuller test is in Figure 14.

```
#Decomposing time series into 3 main components: trend, seasonality and residuals for freight and mail traffic

from statsmodels.tsa.seasonal import seasonal_decompose

plt.rcParams['figure.figsize'] = 10, 5

# Additive decomposition
fnm_decomposed_add = seasonal_decompose(df_FNM, model="additive")
fnm_add = fnm_decomposed_add.plot()
plt.show()
```

Figure 12: Graph for additive decomposition

```
# Multiplicative decomposition
decomposed_mul = seasonal_decompose(df_FNM, model="multiplicative")
mul = decomposed_mul.plot()
plt.show()
```

Figure 13: Graph for multiplicative decomposition

Stationarity Test

```
def stationarity_test(timeseries):

   #Getting the rolling statistics for window = 6 that is next 6 months statistics
    rolling_mean = timeseries.rolling(window = 6).mean()
    rolling_std = timeseries.rolling(window = 6).std()

    #Plotting the rolling statistic
    plt.figure(figsize= (10,6))
    plt.xlabel('Time Period')
    plt.ylabel('Freight and Mail')
    plt.title('Stationary Test: Rolling Mean and Standard Deviation')
    plt.plot(timeseries, color= 'blue', label= 'Original')
    plt.plot(rolling_mean, color= 'green', label= 'Rolling Mean')
    plt.plot(rolling_std, color= 'red', label= 'Rolling Std')
    plt.legend()
    plt.show()

    #Dickey-Fuller test
    #1. Null Hypothesis (H0): if failed to be rejected (high p-value) means it is non-stationary
    #2. Null Hypothesis (H1): if H0 is rejected (low p-value) means it is stationary
    print('Results of Dickey-Fuller Test')
    df_test = adfuller(timeseries)
    df_output = pd.Series(df_test[0:4], index = ['Test Statistic', 'p-value', '#Lags Used', 'Number of Observations Used'])
    for key, value in df_test[4].items():
        df_output['Critical Value (%s)' %key] = value
    print(df_output)
```

```
# Testing the stationarity score for freight and mail of top 5 airports
stationarity_test(df_FNM)
```

Figure 14: Stationarity test

The next step involves the conversion of non-stationary data to stationary data using the differencing method in Figure 15. Other techniques such as log transformation, moving average and weighted moving average are also implemented as a trial to obtain the stationarity test values.

Converting Non Stationary Data to Stationary

```
#Differencing first order the time series to obtain stationarity for air passenger

df_fnm_diff = df_fnm.diff()  # First order differencing
df_fnm_diff = df_fnm_diff.dropna()
plt.xlabel('Time Period')
plt.ylabel('Freight and Mail')
plt.title('Converting Non Stationary Data to Stationary Data using Differencing')
plt.plot(df_fnm_diff)
```

Figure 15: Stationarity test

Figure 16 shows the code for plotting ACF and PACF plots to identify lags.

```
# #ACF and PACF graphs for freight and mail traffic

from pandas.plotting import autocorrelation_plot
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

fig,ax = plt.subplots(2,1,figsize=(15,8))
plot_acf(df_FNM, lags=19, ax=ax[0])
plot_pacf(df_FNM, lags=19, ax=ax[1])
plt.show()
```

Figure 16: ACF and PACF plots

Freight data is now split into training and testing sets. In Figure 17 data from 2020 to 2022 is taken as train data and from 2023 it is taken as test data.

5 Data Modelling and Evaluation

After data is split, Figure 18 shows the auto arima library which is imported to build and fit the model with the best parameters selected automatically by the auto arima model.

In Figure 19 diagnostic plot for the obtained SARIMA model with automatic parameters selected is displayed.

After the auto arima model values are obtained, the model is fit on the length of the

Training and Testing Data Split

```
# Split into training and testing for freight and mail
df_fnm_train = df_FNM.loc[:'2022']
df_fnm_test = df_FNM.loc['2023':]

# Plot the last 3 years of training data and the 6 months of testing
ax = df_fnm_train[-12*3:].plot(figsize=(10, 5))
df_fnm_test.plot(ax=ax)
plt.legend(['Train', 'Test'])
plt.xlabel('Time Period')
plt.ylabel('Freight and Mail')
plt.show()
```

Figure 17: Training and testing split

```
# Build and fit the model

from pmdarima.arima import auto_arima
AUTO_ARIMA_fnm = auto_arima(df_fnm_train, seasonal = True, m = 12, suppress_warnings = True)
AUTO_ARIMA_model_fit = AUTO_ARIMA_fnm.fit(df_fnm_train)
print(AUTO_ARIMA_model_fit.summary())
```

Figure 18: ARIMA/SARIMA

```
#Diagnostics Plot - SARIMA for FNM traffic

plt.rcParams['figure.figsize'] = 18, 14
plot = AUTO_ARIMA_model_fit.plot_diagnostics()
plt.show()
```

Figure 19: Diagnostic plots

test data to make predictions. In Figure 20 the code also includes the plot for actual vs. predicted values.

```
# Build and fit the model

from pmdarima.arima import auto_arima
AUTO_ARIMA_fnm = auto_arima(df_fnm_train, seasonal = True, m = 12, suppress_warnings = True)
AUTO_ARIMA_model_fit = AUTO_ARIMA_fnm.fit(df_fnm_train)
print(AUTO_ARIMA_model_fit.summary())
```

Figure 20: Model fitted for predictions

Figure 21 displays the evaluation metrics for the model.

```
#FNM traffic evaluation metrics

# Predicted values and actual values
auto_p_fnm_values = forecast_diff.values
auto_actual_fnm_values = df_fnm_test.values.flatten()

# Mean Absolute Error (MAE)
mae_fnm_auto = np.mean(np.abs(auto_p_fnm_values - auto_actual_fnm_values))
print("autoARIMA MAE:", mae_fnm_auto)

# Mean Squared Error (MSE)
mse_fnm_auto = np.mean((auto_p_fnm_values - auto_actual_fnm_values) ** 2)
print("autoARIMA MSE:", mse_fnm_auto)

# Root Mean Squared Error (RMSE)
rmse_fnm_auto = np.sqrt(mse_fnm_auto)
print("autoARIMA RMSE:", rmse_fnm_auto)
```

Figure 21: Evaluation metrics

Similarly, three other models are implemented. Figure 22 shows simple exponential smoothing model. Figure 23 shows double exponential smoothing model or Holt's Linear and Figure 24 shows triple exponential smoothing model or Holt-Winters.

The code which helps to answer the research question and objectives is in Figure 25. All the time series models are compared against the actual data.

Figure 26 shows the evaluation metrics table code demand forecast.

The predicted values for January 2023 to June 2023 are in Figure 27.

```python
#Fitting Simple Exponential Smoothing Model for freight and mail traffic
from statsmodels.tsa.holtwinters import SimpleExpSmoothing
simple_model_fnm = SimpleExpSmoothing(df_fnm_train)
fit_simple_fnm = simple_model_fnm.fit()

#Forecast using fitted model
forecast_simple_fnm = fit_simple_fnm.forecast(len(df_fnm_test))
```

```python
#Calculating Evaluation Metrics
mse_se = mean_squared_error(df_fnm_test, forecast_simple_fnm)
rmse_se = np.sqrt(mse_se)
mae_se = mean_absolute_error(df_fnm_test, forecast_simple_fnm)

print('SES MSE:', mse_se)
print('SES RMSE:', rmse_se)
print('SES MAE:', mae_se)
```

```python
# Plotting forecast with training data
plt.figure(figsize=(10, 6))

# Plotting the actual and predicted values
plt.plot(df_fnm_test, label='Actual', marker='o', linestyle='-', color='blue')
plt.plot(forecast_simple_fnm, label='Predicted', marker='o', linestyle='-', color='orange')

# Adding labels and title
plt.xlabel('Time Period')
plt.ylabel('FNM Count')
plt.title('Actual vs. Predicted')
plt.legend()

# Adding grid for better readability
plt.grid(True, linestyle='--', alpha=0.7)

# Displaying the plot
plt.show()
```

Figure 22: Simple Exponential Smoothing

```
#Fitting the Holt's Linear Exponential Smoothing model for freight and mail traffic
from statsmodels.tsa.holtwinters import ExponentialSmoothing
HL_model = ExponentialSmoothing(df_fnm_train, trend='add', damped_trend=True)
fit_HL_fnm = HL_model.fit()

# Forecast using the fitted model
forecast_HL_fnm = fit_HL_fnm.forecast(len(df_fnm_test))
```

```
#Calculating Evaluation Metrics
mse_HL = mean_squared_error(df_fnm_test, forecast_HL_fnm)
rmse_HL = np.sqrt(mse_HL)
mae_HL = mean_absolute_error(df_fnm_test, forecast_HL_fnm)

print('HL MSE:', mse_HL)
print('HL RMSE:', rmse_HL)
print('HL MAE:', mae_HL)
```

```
# Plotting forecast with training data
plt.figure(figsize=(10, 6))

# Plotting the actual and predicted values
plt.plot(df_fnm_test, label='Actual', marker='o', linestyle='-', color='blue')
plt.plot(forecast_HL_fnm, label='Predicted', marker='o', linestyle='-', color='orange')

# Adding labels and title
plt.xlabel('Time Period')
plt.ylabel('FNM Count')
plt.title('Actual vs. Predicted')
plt.legend()

# Adding grid for better readability
plt.grid(True, linestyle='--', alpha=0.7)

# Displaying the plot
plt.show()
```

Figure 23: Double Exponential Smoothing

```
#Fitting the Holt-Winters' Exponential Smoothing model for freight and mail traffic
from statsmodels.tsa.holtwinters import ExponentialSmoothing
HW_model = ExponentialSmoothing(df_fnm_train, trend = 'add', seasonal = 'add', seasonal_periods = 12)
fit_HW_fnm = HW_model.fit()

# Forecast using the fitted model
forecast_HW_fnm = fit_HW_fnm.forecast(len(df_fnm_test))
```

```
#Calculating Evaluation Metrics
mse_HW = mean_squared_error(df_fnm_test, forecast_HW_fnm)
rmse_HW = np.sqrt(mse_HW)
mae_HW = mean_absolute_error(df_fnm_test, forecast_HW_fnm)

print('HW MSE:', mse_HW)
print('HW RMSE:', rmse_HW)
print('HW MAE:', mae_HW)
```

```
# Plotting forecast with training data
plt.figure(figsize=(10, 6))

# Plotting the actual and predicted values
plt.plot(df_fnm_test, label='Actual', marker='o', linestyle='-', color='blue')
plt.plot(forecast_HW_fnm, label='Predicted', marker='o', linestyle='-', color='orange')

# Adding labels and title
plt.xlabel('Time Period')
plt.ylabel('FNM Count')
plt.title('Actual vs. Predicted')
plt.legend()

# Adding grid for better readability
plt.grid(True, linestyle='--', alpha=0.7)

# Displaying the plot
plt.show()
```

Figure 24: Triple Exponential Smoothing

```
# Plotting only the predicted values
plt.figure(figsize=(12, 6))
plt.plot(df_FNM.index[-len(df_fnm_test):], df_FNM[-len(df_fnm_test):], label='Actual', marker='o', linestyle='-', color='blue')
plt.plot(df_fnm_test.index, forecast_diff, label='SARIMA Predicted', marker='o', linestyle='-', color='yellow')
plt.plot(df_fnm_test.index, forecast_simple_fnm, label='SES Predicted', marker='o', linestyle='-', color='orange')
plt.plot(df_fnm_test.index, forecast_HL_fnm, label="Holt's Predicted", marker='o', linestyle='-', color='green')
plt.plot(df_fnm_test.index, forecast_HW_fnm, label='Holt-Winters Predicted', marker='o', linestyle='-', color='red')

plt.xlabel('Time Period')
plt.ylabel('Passenger Count')
plt.title('Comparison of Forecasting Models: SARIMA, SES, Holt, Holt-Winters')
plt.legend()
plt.grid(True, linestyle='--', alpha=0.7)
plt.show()
```

Figure 25: Graph for model comparison

```
#Creating table for evaluation metrics
from tabulate import tabulate

# Store evaluation metrics in a dictionary
eval_metrics_dict = {
    'SARIMA': {'MAE': mae_fnm_auto, 'MSE': mse_fnm_auto, 'RMSE': rmse_fnm_auto},
    'SES': {'MAE': mae_se, 'MSE': mse_se, 'RMSE': rmse_se},
    'Holt Linear': {'MAE': mae_HL, 'MSE': mse_HL, 'RMSE': rmse_HL},
    'Holt-Winters': {'MAE': mae_HW, 'MSE': mse_HW, 'RMSE': rmse_HW},
}

# Find the model with the lowest RMSE
best_model_fnm = min(eval_metrics_dict, key=lambda k: eval_metrics_dict[k]['RMSE'])

# Convert evaluation metrics to a list of lists
table_data = [["Model", "MAE", "MSE", "RMSE"]]
for model, metrics in eval_metrics_dict.items():
    table_data.append([model, f"{metrics['MAE']:.2f}", f"{metrics['MSE']:.2f}", f"{metrics['RMSE']:.2f}"])

# Print the table using the tabulate module
print(tabulate(table_data, headers="firstrow"))

# Print the best model
print(f"\nThe best model is: {best_model_fnm} with RMSE: {eval_metrics_dict[best_model_fnm]['RMSE']:.2f}")
```

Figure 26: Evaluation metrics table for model comparison

```
# Combine actual and predicted values into a DataFrame
results_fnm = pd.DataFrame({
    'Time Period': df_fnm_test.index,
    'Actual': df_fnm_test['OBS_VALUE'].values,
    'SES Predicted': forecast_simple_fnm.values,
    "Holt's Predicted": forecast_HL_fnm.values,
    'Holt-Winters Predicted': forecast_HW_fnm.values,
    'SARIMA Predicted': forecast_diff.values
})
# Find the model with the lowest RMSE
best_model_fnm = min(eval_metrics_dict, key=lambda k: eval_metrics_dict[k]['RMSE'])

# Set the 'Time Period' column as the index
results_fnm.set_index('Time Period', inplace=True)

# Display the results DataFrame
print(results_fnm)
```

Figure 27: Predicted values for demand forecast