**Practical Report on Machine Learning Project**

**1. Name of the Data**  
Airline Passengers Forecasting Dataset

**2. Source of the Data**  
This dataset originates from Kaggle Datasets.

**3. Link to the Original Data**  
<https://www.kaggle.com/datasets/oezlem/airline-passenger-forecasting?select=airline-passengers.csv>

**4. Data Explanation**  
The dataset contains monthly totals of international airline passengers from 1949 to 1960.  
It has two columns: Month and the number of passengers.  
To use it with machine learning models, the data was transformed using a sliding window technique to generate lag-based features, converting it into a supervised learning format.

**5. Type of Problem**  
Time series forecasting and regression

**6. Number of Attributes**  
Several lag-based features were engineered from the original "Passengers" column.

**7. Number of Samples**  
144 monthly observations, later converted into training samples using a windowing approach.

**8. Properties of the Data (Statistics)**

* Minimum number of passengers: 104
* Maximum number of passengers: 622
* Mean: Approximately 280
* Standard Deviation: Approximately 119

**9. Missing Data**  
No missing data is present in the dataset.

**10. Data Visualization**  
Line plots were used to observe trends and seasonality in the passenger numbers over time.

**11. Normalization or Standardization**  
Normalization was applied because some models (e.g., SVM, KNN, and ANN) are sensitive to feature scales.  
StandardScaler was used to standardize the features for consistency across models.

**12. Preprocessing Applied**

* Parsed 'Month' column as datetime
* Set 'Month' as index
* Generated lag features
* Applied feature scaling
* Split data into train and test sets

**13. Train-Test Split**  
Data was split using an 80/20 ratio based on chronological order to maintain the time series nature.

**14. Machine Learning Models and Performance**

* **Classical Time Series Models:**
  + Simple Exponential Smoothing (SES)
  + Holt-Winters Exponential Smoothing
  + ARIMA
  + SARIMA
  + SARIMAX
* Machine Learning Models:  
    
  - Linear Regression  
  - Decision Tree  
  - K-Nearest Neighbors (KNN)  
  - Naive Bayes  
  - Artificial Neural Network (ANN)  
  - Support Vector Machine (SVM)  
  - Random Forest Regressor
  + Support Vector Machine (SVM)
  + Random Forest (RF)
  + Decision Tree
  + K-Nearest Neighbors (KNN)
  + Naive Bayes
  + Linear Regression

**15. Accuracy and Figures**  
Models were evaluated using Mean Absolute Error (MAE).  
Prediction performance was visualized by plotting the true values against the predicted values.

**16. Advanced Visualization**  
Error distributions and performance curves were plotted using Matplotlib to compare model behaviors across classical and ML models.

**18. Project Structure Note**  
The code is located in the main folder under the name main.ipynb.  
The dataset is stored in the Data folder, which includes:

* original data for raw files
* preprocessed data for train/test sets
* Results for output from all models

Link of project in github :

<https://github.com/salALotaibi2/Airline_passenger_forecasting->

**Importing Libraries**

In [1]:

**import** itertools

**import** warnings

**import** numpy **as** np

**import** pandas **as** pd

**from** matplotlib **import** pyplot **as** plt

**import** plotly.express **as** px

**import** statsmodels.api **as** sm

**import** statsmodels.tsa.api **as** smt

**from** statsmodels.tsa.arima\_model **import** ARIMA

**from** statsmodels.tsa.seasonal **import** seasonal\_decompose

**from** statsmodels.tsa.holtwinters **import** SimpleExpSmoothing

**from** statsmodels.tsa.holtwinters **import** ExponentialSmoothing

**from** statsmodels.tsa.statespace.sarimax **import** SARIMAX

**from** sklearn.metrics **import** mean\_absolute\_error

warnings**.**filterwarnings('ignore')

**Loading the Dataset**

In [2]:

df1 **=** pd**.**read\_csv("Data/airline-passengers.csv", index\_col**=**'Month', parse\_dates**=True**)

df **=** df1**.**copy()

df**.**head()

Out[2]:

|  | **total\_passengers** |
| --- | --- |
| **Month** |  |
| **1949-01-01** | 112 |
| **1949-02-01** | 118 |
| **1949-03-01** | 132 |
| **1949-04-01** | 129 |
| **1949-05-01** | 121 |

**Exploratory Data Analysis**

In [3]:

**def** check\_df(dataframe, head**=**10):

print('\033[1m' **+** 20**\***"\*" **+** ' SHAPE ' **+** 20**\***"\*" **+** '\033[0m')

print(dataframe**.**shape)

print('\033[1m' **+** 20**\***"\*" **+** ' TYPES ' **+** 20**\***"\*" **+** '\033[0m')

print(dataframe**.**dtypes)

print('\033[1m' **+** 20**\***"\*" **+** ' NA ' **+** 20**\***"\*" **+** '\033[0m')

print(dataframe**.**isnull()**.**sum())

print('\033[1m' **+** 20**\***"\*" **+** ' TAIL ' **+** 20**\***"\*" **+** '\033[0m')

print(dataframe**.**tail())

check\_df(df,10)

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* SHAPE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

(144, 1)

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* TYPES \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

total\_passengers int64

dtype: object

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* NA \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

total\_passengers 0

dtype: int64

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* TAIL \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

total\_passengers

Month

1960-08-01 606

1960-09-01 508

1960-10-01 461

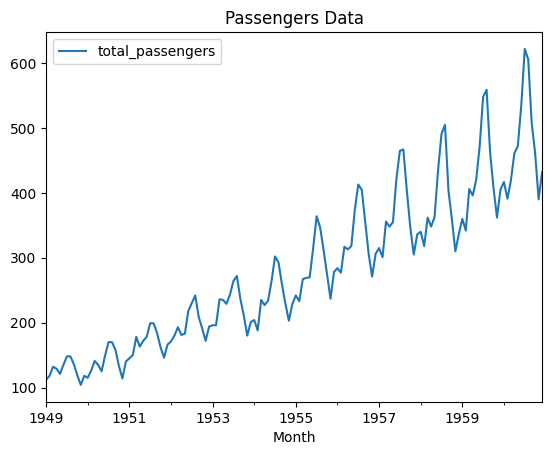
1960-11-01 390

1960-12-01 432

In [4]:

df[['total\_passengers']]**.**plot(title**=**'Passengers Data')

plt**.**show()



When we examined the time series, we could see trend and seasonality. The series is not stationary.

The level can be understood as the average value of the data point in time-series data. The trend means an increasing or decreasing value in time-series data. Seasonality means repeating the pattern of a cycle in the time-series data. Noise means random variance in time-series data.

In [5]:

df**.**index

Out[5]:

DatetimeIndex(['1949-01-01', '1949-02-01', '1949-03-01', '1949-04-01',

'1949-05-01', '1949-06-01', '1949-07-01', '1949-08-01',

'1949-09-01', '1949-10-01',

...

'1960-03-01', '1960-04-01', '1960-05-01', '1960-06-01',

'1960-07-01', '1960-08-01', '1960-09-01', '1960-10-01',

'1960-11-01', '1960-12-01'],

dtype='datetime64[ns]', name='Month', length=144, freq=None)

In [6]:

df**.**index**.**freq **=** "MS"

df**.**index

Out[6]:

DatetimeIndex(['1949-01-01', '1949-02-01', '1949-03-01', '1949-04-01',

'1949-05-01', '1949-06-01', '1949-07-01', '1949-08-01',

'1949-09-01', '1949-10-01',

...

'1960-03-01', '1960-04-01', '1960-05-01', '1960-06-01',

'1960-07-01', '1960-08-01', '1960-09-01', '1960-10-01',

'1960-11-01', '1960-12-01'],

dtype='datetime64[ns]', name='Month', length=144, freq='MS')

**Holdout Method**

In [7]:

train **=** df[:120]

test **=** df[120:]

len(train)

len(test)

**Single Exponential Smoothing (SES)**

In [ ]:

In [8]:

**def** ses\_optimizer(train, alphas, step**=**48):

best\_alpha, best\_mae **=** **None**, float("inf")

**for** alpha **in** alphas:

ses\_model **=** SimpleExpSmoothing(train)**.**fit(smoothing\_level**=**alpha)

y\_pred **=** ses\_model**.**forecast(step)

mae **=** mean\_absolute\_error(test, y\_pred)

**if** mae **<** best\_mae:

best\_alpha, best\_mae **=** alpha, mae

print("alpha:", round(alpha, 2), "mae:", round(mae, 4))

print("best\_alpha:", round(best\_alpha, 2), "best\_mae:", round(best\_mae, 4))

**return** best\_alpha, best\_mae

In [9]:

alphas **=** np**.**arange(0.01, 1, 0.10)

best\_alpha, best\_mae **=** ses\_optimizer(train, alphas, step**=**24)

alpha: 0.01 mae: 225.5863

alpha: 0.11 mae: 82.528

alpha: 0.21 mae: 82.8979

alpha: 0.31 mae: 89.8377

alpha: 0.41 mae: 99.0585

alpha: 0.51 mae: 107.5558

alpha: 0.61 mae: 113.7514

alpha: 0.71 mae: 117.2224

alpha: 0.81 mae: 118.1776

alpha: 0.91 mae: 117.2438

best\_alpha: 0.11 best\_mae: 82.528

In [10]:

ses\_model **=** SimpleExpSmoothing(train)**.**fit(smoothing\_level**=**best\_alpha)

In [11]:

y\_pred **=** ses\_model**.**forecast(24)

In [12]:

**def** plot\_prediction(y\_pred, label):

train["total\_passengers"]**.**plot(legend**=True**, label**=**"TRAIN")

test["total\_passengers"]**.**plot(legend**=True**, label**=**"TEST")

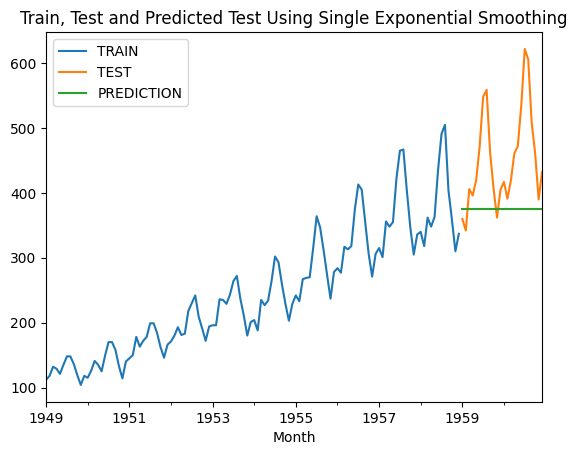
y\_pred**.**plot(legend**=True**, label**=**"PREDICTION")

plt**.**title("Train, Test and Predicted Test Using "**+**label)

plt**.**show()

In [13]:

plot\_prediction(y\_pred, "Single Exponential Smoothing")



SES could not make a successful prediction because there was seasonality and trend in the time series.

**Double Exponential Smoothing (DES)**

In [14]:

**def** des\_optimizer(train, alphas, betas, step**=**48):

best\_alpha, best\_beta, best\_mae **=** **None**, **None**, float("inf")

**for** alpha **in** alphas:

**for** beta **in** betas:

des\_model **=** ExponentialSmoothing(train, trend**=**"add")**.**fit(smoothing\_level**=**alpha, smoothing\_slope**=**beta)

y\_pred **=** des\_model**.**forecast(step)

mae **=** mean\_absolute\_error(test, y\_pred)

**if** mae **<** best\_mae:

best\_alpha, best\_beta, best\_mae **=** alpha, beta, mae

print("alpha:", round(alpha, 2), "beta:", round(beta, 2), "mae:", round(mae, 4))

print("best\_alpha:", round(best\_alpha, 2), "best\_beta:", round(best\_beta, 2), "best\_mae:", round(best\_mae, 4))

**return** best\_alpha, best\_beta, best\_mae

In [15]:

alphas **=** np**.**arange(0.01, 1, 0.10)

betas **=** np**.**arange(0.01, 1, 0.10)

In [16]:

best\_alpha, best\_beta, best\_mae **=** des\_optimizer(train, alphas, betas, step**=**24)

alpha: 0.01 beta: 0.01 mae: 54.9512

alpha: 0.01 beta: 0.11 mae: 54.1036

alpha: 0.01 beta: 0.21 mae: 55.5569

alpha: 0.01 beta: 0.31 mae: 57.3247

alpha: 0.01 beta: 0.41 mae: 57.7931

alpha: 0.01 beta: 0.51 mae: 57.9968

alpha: 0.01 beta: 0.61 mae: 57.9637

alpha: 0.01 beta: 0.71 mae: 57.5635

alpha: 0.01 beta: 0.81 mae: 57.3334

alpha: 0.01 beta: 0.91 mae: 57.8984

alpha: 0.11 beta: 0.01 mae: 55.0309

alpha: 0.11 beta: 0.11 mae: 58.4728

alpha: 0.11 beta: 0.21 mae: 69.6339

alpha: 0.11 beta: 0.31 mae: 72.555

alpha: 0.11 beta: 0.41 mae: 78.8168

alpha: 0.11 beta: 0.51 mae: 81.996

alpha: 0.11 beta: 0.61 mae: 77.2262

alpha: 0.11 beta: 0.71 mae: 71.5042

alpha: 0.11 beta: 0.81 mae: 69.0258

alpha: 0.11 beta: 0.91 mae: 67.8408

alpha: 0.21 beta: 0.01 mae: 57.1824

alpha: 0.21 beta: 0.11 mae: 74.992

alpha: 0.21 beta: 0.21 mae: 95.9492

alpha: 0.21 beta: 0.31 mae: 115.3164

alpha: 0.21 beta: 0.41 mae: 139.4303

alpha: 0.21 beta: 0.51 mae: 178.3759

alpha: 0.21 beta: 0.61 mae: 243.4543

alpha: 0.21 beta: 0.71 mae: 345.299

alpha: 0.21 beta: 0.81 mae: 489.8453

alpha: 0.21 beta: 0.91 mae: 675.8334

alpha: 0.31 beta: 0.01 mae: 65.3308

alpha: 0.31 beta: 0.11 mae: 102.5159

alpha: 0.31 beta: 0.21 mae: 150.3694

alpha: 0.31 beta: 0.31 mae: 213.3573

alpha: 0.31 beta: 0.41 mae: 307.2251

alpha: 0.31 beta: 0.51 mae: 441.9762

alpha: 0.31 beta: 0.61 mae: 608.1276

alpha: 0.31 beta: 0.71 mae: 778.9847

alpha: 0.31 beta: 0.81 mae: 921.2448

alpha: 0.31 beta: 0.91 mae: 1013.8071

alpha: 0.41 beta: 0.01 mae: 74.6629

alpha: 0.41 beta: 0.11 mae: 134.9652

alpha: 0.41 beta: 0.21 mae: 214.0271

alpha: 0.41 beta: 0.31 mae: 321.1816

alpha: 0.41 beta: 0.41 mae: 457.7845

alpha: 0.41 beta: 0.51 mae: 604.528

alpha: 0.41 beta: 0.61 mae: 731.2542

alpha: 0.41 beta: 0.71 mae: 816.4681

alpha: 0.41 beta: 0.81 mae: 856.8799

alpha: 0.41 beta: 0.91 mae: 861.1779

alpha: 0.51 beta: 0.01 mae: 83.8758

alpha: 0.51 beta: 0.11 mae: 162.7106

alpha: 0.51 beta: 0.21 mae: 262.9556

alpha: 0.51 beta: 0.31 mae: 384.168

alpha: 0.51 beta: 0.41 mae: 510.8847

alpha: 0.51 beta: 0.51 mae: 615.718

alpha: 0.51 beta: 0.61 mae: 679.6289

alpha: 0.51 beta: 0.71 mae: 699.5586

alpha: 0.51 beta: 0.81 mae: 681.8415

alpha: 0.51 beta: 0.91 mae: 633.8713

alpha: 0.61 beta: 0.01 mae: 91.1494

alpha: 0.61 beta: 0.11 mae: 180.5749

alpha: 0.61 beta: 0.21 mae: 286.5276

alpha: 0.61 beta: 0.31 mae: 398.7627

alpha: 0.61 beta: 0.41 mae: 494.8066

alpha: 0.61 beta: 0.51 mae: 552.8904

alpha: 0.61 beta: 0.61 mae: 565.2675

alpha: 0.61 beta: 0.71 mae: 535.41

alpha: 0.61 beta: 0.81 mae: 470.4302

alpha: 0.61 beta: 0.91 mae: 377.2283

alpha: 0.71 beta: 0.01 mae: 95.5898

alpha: 0.71 beta: 0.11 mae: 188.0569

alpha: 0.71 beta: 0.21 mae: 289.0405

alpha: 0.71 beta: 0.31 mae: 381.6533

alpha: 0.71 beta: 0.41 mae: 443.8336

alpha: 0.71 beta: 0.51 mae: 461.4037

alpha: 0.71 beta: 0.61 mae: 433.7736

alpha: 0.71 beta: 0.71 mae: 368.1872

alpha: 0.71 beta: 0.81 mae: 274.2863

alpha: 0.71 beta: 0.91 mae: 162.1223

alpha: 0.81 beta: 0.01 mae: 97.0247

alpha: 0.81 beta: 0.11 mae: 187.5378

alpha: 0.81 beta: 0.21 mae: 277.7663

alpha: 0.81 beta: 0.31 mae: 348.1264

alpha: 0.81 beta: 0.41 mae: 380.2822

alpha: 0.81 beta: 0.51 mae: 367.0635

alpha: 0.81 beta: 0.61 mae: 313.3918

alpha: 0.81 beta: 0.71 mae: 230.0239

alpha: 0.81 beta: 0.81 mae: 129.043

alpha: 0.81 beta: 0.91 mae: 60.7209

alpha: 0.91 beta: 0.01 mae: 96.16

alpha: 0.91 beta: 0.11 mae: 181.6748

alpha: 0.91 beta: 0.21 mae: 258.9646

alpha: 0.91 beta: 0.31 mae: 308.5961

alpha: 0.91 beta: 0.41 mae: 317.1132

alpha: 0.91 beta: 0.51 mae: 283.2875

alpha: 0.91 beta: 0.61 mae: 216.0383

alpha: 0.91 beta: 0.71 mae: 127.9546

alpha: 0.91 beta: 0.81 mae: 59.6742

alpha: 0.91 beta: 0.91 mae: 94.305

best\_alpha: 0.01 best\_beta: 0.11 best\_mae: 54.1036

In [17]:

des\_model **=** ExponentialSmoothing(train, trend**=**"add")**.**fit(smoothing\_level**=**best\_alpha,

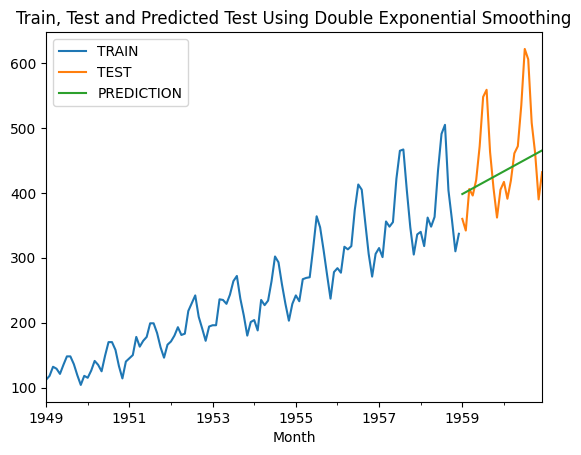
smoothing\_slope**=**best\_beta)

In [18]:

y\_pred **=** des\_model**.**forecast(24)

In [19]:

plot\_prediction(y\_pred, "Double Exponential Smoothing")



It caught the trend in the time series, but could not catch the seasonality.

**Triple Exponential Smoothing (TES) (Holt-Winters)**

In [20]:

**def** tes\_optimizer(train, abg, step**=**48):

best\_alpha, best\_beta, best\_gamma, best\_mae **=** **None**, **None**, **None**, float("inf")

**for** comb **in** abg:

tes\_model **=** ExponentialSmoothing(train, trend**=**"add", seasonal**=**"add", seasonal\_periods**=**12)**.**\

fit(smoothing\_level**=**comb[0], smoothing\_slope**=**comb[1], smoothing\_seasonal**=**comb[2])

y\_pred **=** tes\_model**.**forecast(step)

mae **=** mean\_absolute\_error(test, y\_pred)

**if** mae **<** best\_mae:

best\_alpha, best\_beta, best\_gamma, best\_mae **=** comb[0], comb[1], comb[2], mae

print([round(comb[0], 2), round(comb[1], 2), round(comb[2], 2), round(mae, 2)])

print("best\_alpha:", round(best\_alpha, 2), "best\_beta:", round(best\_beta, 2), "best\_gamma:", round(best\_gamma, 2),

"best\_mae:", round(best\_mae, 4))

**return** best\_alpha, best\_beta, best\_gamma, best\_mae

In [21]:

alphas **=** betas **=** gammas **=** np**.**arange(0.10, 1, 0.20)

abg **=** list(itertools**.**product(alphas, betas, gammas))

In [22]:

best\_alpha, best\_beta, best\_gamma, best\_mae **=** tes\_optimizer(train, abg, step**=**24)

[0.1, 0.1, 0.1, 36.83]

[0.1, 0.1, 0.3, 34.88]

[0.1, 0.1, 0.5, 35.91]

[0.1, 0.1, 0.7, 38.72]

[0.1, 0.1, 0.9, 42.55]

[0.1, 0.3, 0.1, 53.96]

[0.1, 0.3, 0.3, 53.0]

[0.1, 0.3, 0.5, 56.98]

[0.1, 0.3, 0.7, 61.38]

[0.1, 0.3, 0.9, 67.17]

[0.1, 0.5, 0.1, 56.76]

[0.1, 0.5, 0.3, 51.72]

[0.1, 0.5, 0.5, 53.39]

[0.1, 0.5, 0.7, 60.08]

[0.1, 0.5, 0.9, 79.29]

[0.1, 0.7, 0.1, 34.04]

[0.1, 0.7, 0.3, 25.93]

[0.1, 0.7, 0.5, 27.81]

[0.1, 0.7, 0.7, 27.91]

[0.1, 0.7, 0.9, 38.32]

[0.1, 0.9, 0.1, 26.87]

[0.1, 0.9, 0.3, 25.32]

[0.1, 0.9, 0.5, 53.88]

[0.1, 0.9, 0.7, 38.38]

[0.1, 0.9, 0.9, 17.78]

[0.3, 0.1, 0.1, 54.54]

[0.3, 0.1, 0.3, 38.34]

[0.3, 0.1, 0.5, 31.57]

[0.3, 0.1, 0.7, 30.99]

[0.3, 0.1, 0.9, 29.23]

[0.3, 0.3, 0.1, 72.71]

[0.3, 0.3, 0.3, 22.6]

[0.3, 0.3, 0.5, 11.99]

[0.3, 0.3, 0.7, 17.11]

[0.3, 0.3, 0.9, 22.32]

[0.3, 0.5, 0.1, 132.33]

[0.3, 0.5, 0.3, 25.74]

[0.3, 0.5, 0.5, 17.3]

[0.3, 0.5, 0.7, 38.38]

[0.3, 0.5, 0.9, 56.82]

[0.3, 0.7, 0.1, 288.32]

[0.3, 0.7, 0.3, 134.86]

[0.3, 0.7, 0.5, 96.77]

[0.3, 0.7, 0.7, 95.55]

[0.3, 0.7, 0.9, 114.59]

[0.3, 0.9, 0.1, 452.01]

[0.3, 0.9, 0.3, 274.71]

[0.3, 0.9, 0.5, 185.56]

[0.3, 0.9, 0.7, 158.95]

[0.3, 0.9, 0.9, 186.84]

[0.5, 0.1, 0.1, 82.19]

[0.5, 0.1, 0.3, 55.63]

[0.5, 0.1, 0.5, 34.87]

[0.5, 0.1, 0.7, 26.03]

[0.5, 0.1, 0.9, 26.71]

[0.5, 0.3, 0.1, 177.09]

[0.5, 0.3, 0.3, 113.12]

[0.5, 0.3, 0.5, 60.59]

[0.5, 0.3, 0.7, 40.88]

[0.5, 0.3, 0.9, 39.6]

[0.5, 0.5, 0.1, 316.38]

[0.5, 0.5, 0.3, 248.61]

[0.5, 0.5, 0.5, 165.44]

[0.5, 0.5, 0.7, 102.0]

[0.5, 0.5, 0.9, 75.54]

[0.5, 0.7, 0.1, 412.24]

[0.5, 0.7, 0.3, 366.81]

[0.5, 0.7, 0.5, 258.84]

[0.5, 0.7, 0.7, 158.39]

[0.5, 0.7, 0.9, 128.09]

[0.5, 0.9, 0.1, 438.76]

[0.5, 0.9, 0.3, 441.23]

[0.5, 0.9, 0.5, 323.59]

[0.5, 0.9, 0.7, 229.22]

[0.5, 0.9, 0.9, 176.77]

[0.7, 0.1, 0.1, 104.36]

[0.7, 0.1, 0.3, 89.35]

[0.7, 0.1, 0.5, 62.87]

[0.7, 0.1, 0.7, 42.14]

[0.7, 0.1, 0.9, 31.43]

[0.7, 0.3, 0.1, 216.29]

[0.7, 0.3, 0.3, 205.22]

[0.7, 0.3, 0.5, 159.22]

[0.7, 0.3, 0.7, 112.57]

[0.7, 0.3, 0.9, 87.28]

[0.7, 0.5, 0.1, 302.42]

[0.7, 0.5, 0.3, 326.36]

[0.7, 0.5, 0.5, 277.99]

[0.7, 0.5, 0.7, 214.6]

[0.7, 0.5, 0.9, 184.66]

[0.7, 0.7, 0.1, 309.0]

[0.7, 0.7, 0.3, 377.0]

[0.7, 0.7, 0.5, 327.37]

[0.7, 0.7, 0.7, 282.74]

[0.7, 0.7, 0.9, 244.25]

[0.7, 0.9, 0.1, 249.12]

[0.7, 0.9, 0.3, 353.99]

[0.7, 0.9, 0.5, 314.03]

[0.7, 0.9, 0.7, 305.42]

[0.7, 0.9, 0.9, 265.45]

[0.9, 0.1, 0.1, 112.57]

[0.9, 0.1, 0.3, 112.84]

[0.9, 0.1, 0.5, 95.59]

[0.9, 0.1, 0.7, 73.88]

[0.9, 0.1, 0.9, 56.35]

[0.9, 0.3, 0.1, 207.83]

[0.9, 0.3, 0.3, 230.8]

[0.9, 0.3, 0.5, 209.17]

[0.9, 0.3, 0.7, 169.62]

[0.9, 0.3, 0.9, 125.11]

[0.9, 0.5, 0.1, 245.78]

[0.9, 0.5, 0.3, 301.42]

[0.9, 0.5, 0.5, 280.73]

[0.9, 0.5, 0.7, 240.08]

[0.9, 0.5, 0.9, 182.98]

[0.9, 0.7, 0.1, 218.45]

[0.9, 0.7, 0.3, 299.86]

[0.9, 0.7, 0.5, 298.5]

[0.9, 0.7, 0.7, 241.07]

[0.9, 0.7, 0.9, 233.79]

[0.9, 0.9, 0.1, 170.16]

[0.9, 0.9, 0.3, 261.35]

[0.9, 0.9, 0.5, 284.25]

[0.9, 0.9, 0.7, 216.63]

[0.9, 0.9, 0.9, 263.57]

best\_alpha: 0.3 best\_beta: 0.3 best\_gamma: 0.5 best\_mae: 11.9947

In [23]:

tes\_model **=** ExponentialSmoothing(train, trend**=**"add", seasonal**=**"add", seasonal\_periods**=**12)**.**\

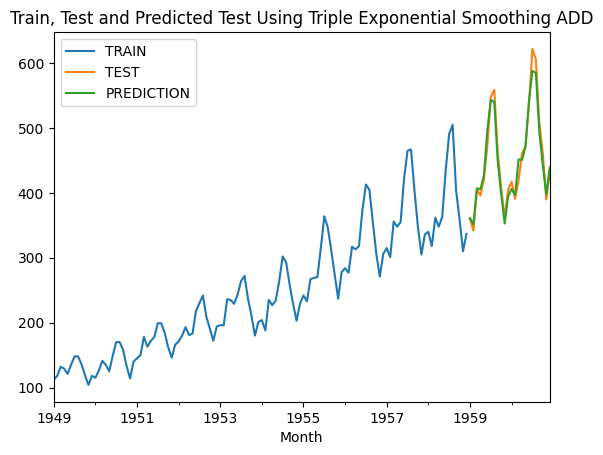
fit(smoothing\_level**=**best\_alpha, smoothing\_slope**=**best\_beta, smoothing\_seasonal**=**best\_gamma)

In [24]:

y\_pred **=** tes\_model**.**forecast(24)

In [25]:

plot\_prediction(y\_pred, "Triple Exponential Smoothing ADD")



We have achieved the best results so far with the Holt Winter - TES method.

**ARIMA(p, d, q): (Autoregressive Integrated Moving Average)**

In [26]:

p **=** d **=** q **=** range(0, 4)

pdq **=** list(itertools**.**product(p, d, q))

In [27]:

**def** arima\_optimizer\_aic(train, orders):

best\_aic, best\_params **=** float("inf"), **None**

**for** order **in** orders:

**try**:

arima\_model **=** sm**.**tsa**.**ARIMA(train, order**=**order)**.**fit()

aic **=** arima\_model**.**aic

**if** aic **<** best\_aic:

best\_aic, best\_params **=** aic, order

print('ARIMA%s AIC=%.2f' **%** (order, aic))

**except**:

**continue**

print('Best ARIMA%s AIC=%.2f' **%** (best\_params, best\_aic))

**return** best\_params

In [28]:

best\_params\_aic **=** arima\_optimizer\_aic(train, pdq)

ARIMA(0, 0, 0) AIC=1436.33

ARIMA(0, 0, 1) AIC=1295.42

ARIMA(0, 0, 2) AIC=1229.22

ARIMA(0, 0, 3) AIC=1169.61

ARIMA(0, 1, 0) AIC=1138.81

ARIMA(0, 1, 1) AIC=1127.02

ARIMA(0, 1, 2) AIC=1126.97

ARIMA(0, 1, 3) AIC=1125.45

ARIMA(0, 2, 0) AIC=1171.65

ARIMA(0, 2, 1) AIC=1136.52

ARIMA(0, 2, 2) AIC=1124.36

ARIMA(0, 2, 3) AIC=1124.57

ARIMA(0, 3, 0) AIC=1263.95

ARIMA(0, 3, 1) AIC=1169.49

ARIMA(0, 3, 2) AIC=1140.39

ARIMA(0, 3, 3) AIC=1127.31

ARIMA(1, 0, 0) AIC=1152.40

ARIMA(1, 0, 1) AIC=1138.85

ARIMA(1, 0, 2) AIC=1140.24

ARIMA(1, 0, 3) AIC=1142.18

ARIMA(1, 1, 0) AIC=1130.66

ARIMA(1, 1, 1) AIC=1125.43

ARIMA(1, 1, 2) AIC=1118.10

ARIMA(1, 1, 3) AIC=1119.79

ARIMA(1, 2, 0) AIC=1168.21

ARIMA(1, 2, 1) AIC=1127.97

ARIMA(1, 2, 2) AIC=1122.89

ARIMA(1, 2, 3) AIC=1123.66

ARIMA(1, 3, 0) AIC=1232.81

ARIMA(1, 3, 1) AIC=1166.48

ARIMA(1, 3, 2) AIC=1170.58

ARIMA(1, 3, 3) AIC=1130.67

ARIMA(2, 0, 0) AIC=1141.99

ARIMA(2, 0, 1) AIC=1138.37

ARIMA(2, 0, 2) AIC=1138.68

ARIMA(2, 0, 3) AIC=1135.89

ARIMA(2, 1, 0) AIC=1126.84

ARIMA(2, 1, 1) AIC=1113.05

ARIMA(2, 1, 2) AIC=1093.07

ARIMA(2, 1, 3) AIC=1119.02

ARIMA(2, 2, 0) AIC=1161.90

ARIMA(2, 2, 1) AIC=1124.41

ARIMA(2, 2, 2) AIC=1123.90

ARIMA(2, 2, 3) AIC=1125.66

ARIMA(2, 3, 0) AIC=1209.77

ARIMA(2, 3, 1) AIC=1160.71

ARIMA(2, 3, 2) AIC=1160.91

ARIMA(2, 3, 3) AIC=1130.92

ARIMA(3, 0, 0) AIC=1139.94

ARIMA(3, 0, 1) AIC=1138.73

ARIMA(3, 0, 2) AIC=1140.56

ARIMA(3, 0, 3) AIC=1107.33

ARIMA(3, 1, 0) AIC=1126.26

ARIMA(3, 1, 1) AIC=1114.44

ARIMA(3, 1, 2) AIC=1112.58

ARIMA(3, 1, 3) AIC=1089.76

ARIMA(3, 2, 0) AIC=1160.89

ARIMA(3, 2, 1) AIC=1123.92

ARIMA(3, 2, 2) AIC=1124.51

ARIMA(3, 2, 3) AIC=1101.67

ARIMA(3, 3, 0) AIC=1208.80

ARIMA(3, 3, 1) AIC=1160.02

ARIMA(3, 3, 2) AIC=1161.12

ARIMA(3, 3, 3) AIC=1130.91

Best ARIMA(3, 1, 3) AIC=1089.76

In [29]:

arima\_model **=** sm**.**tsa**.**ARIMA(train, order**=**best\_params\_aic)**.**fit()

y\_pred **=** arima\_model**.**forecast(steps**=**len(test))

y\_pred **=** pd**.**Series(y\_pred, index**=**test**.**index)

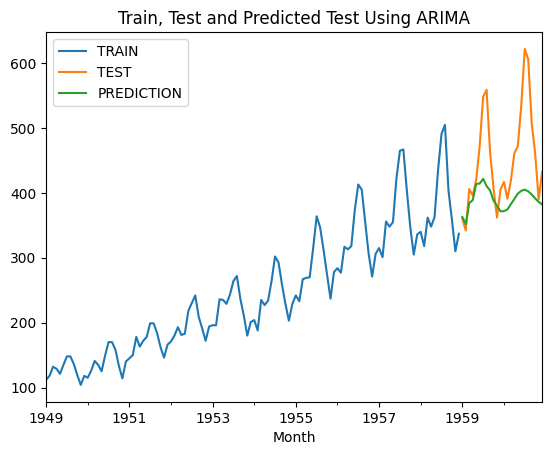
mean\_absolute\_error(test, y\_pred)

Out[29]:

64.01212494280455

In [30]:

plot\_prediction(pd**.**Series(y\_pred, index**=**test**.**index), "ARIMA")



It could not capture the seasonality in the time series.

**SARIMA(p, d, q): (Seasonal Autoregressive Integrated Moving-Average)**

In [31]:

p **=** d **=** q **=** range(0, 2)

pdq **=** list(itertools**.**product(p, d, q))

seasonal\_pdq **=** [(x[0], x[1], x[2], 12) **for** x **in** list(itertools**.**product(p, d, q))]

In [32]:

**def** sarima\_optimizer\_aic(train, pdq, seasonal\_pdq):

best\_aic, best\_order, best\_seasonal\_order **=** float("inf"), float("inf"), **None**

**for** param **in** pdq:

**for** param\_seasonal **in** seasonal\_pdq:

**try**:

sarimax\_model **=** SARIMAX(train, order**=**param, seasonal\_order**=**param\_seasonal)

results **=** sarimax\_model**.**fit(disp**=**0)

aic **=** results**.**aic

**if** aic **<** best\_aic:

best\_aic, best\_order, best\_seasonal\_order **=** aic, param, param\_seasonal

print('SARIMA{}x{}12 - AIC:{}'**.**format(param, param\_seasonal, aic))

**except**:

**continue**

print('SARIMA{}x{}12 - AIC:{}'**.**format(best\_order, best\_seasonal\_order, best\_aic))

**return** best\_order, best\_seasonal\_order

In [33]:

best\_order, best\_seasonal\_order **=** sarima\_optimizer\_aic(train, pdq, seasonal\_pdq)

SARIMA(0, 0, 0)x(0, 0, 0, 12)12 - AIC:1680.2792351899682

SARIMA(0, 0, 0)x(0, 0, 1, 12)12 - AIC:1557.632053117491

SARIMA(0, 0, 0)x(0, 1, 0, 12)12 - AIC:1060.4792883429307

SARIMA(0, 0, 0)x(0, 1, 1, 12)12 - AIC:1014.069674752024

SARIMA(0, 0, 0)x(1, 0, 0, 12)12 - AIC:1231.9865853911651

SARIMA(0, 0, 0)x(1, 0, 1, 12)12 - AIC:1185.301097909221

SARIMA(0, 0, 0)x(1, 1, 0, 12)12 - AIC:969.3781382611646

SARIMA(0, 0, 0)x(1, 1, 1, 12)12 - AIC:951.6074684656228

SARIMA(0, 0, 1)x(0, 0, 0, 12)12 - AIC:1524.3620853403827

SARIMA(0, 0, 1)x(0, 0, 1, 12)12 - AIC:1403.594235049879

SARIMA(0, 0, 1)x(0, 1, 0, 12)12 - AIC:967.522569558382

SARIMA(0, 0, 1)x(0, 1, 1, 12)12 - AIC:942.242509304673

SARIMA(0, 0, 1)x(1, 0, 0, 12)12 - AIC:1136.4998829579354

SARIMA(0, 0, 1)x(1, 0, 1, 12)12 - AIC:1127.8397013046172

SARIMA(0, 0, 1)x(1, 1, 0, 12)12 - AIC:918.223387602213

SARIMA(0, 0, 1)x(1, 1, 1, 12)12 - AIC:896.7245834892016

SARIMA(0, 1, 0)x(0, 0, 0, 12)12 - AIC:1138.8088994229304

SARIMA(0, 1, 0)x(0, 0, 1, 12)12 - AIC:1049.4242820844172

SARIMA(0, 1, 0)x(0, 1, 0, 12)12 - AIC:807.6545889122535

SARIMA(0, 1, 0)x(0, 1, 1, 12)12 - AIC:808.7804147911176

SARIMA(0, 1, 0)x(1, 0, 0, 12)12 - AIC:925.1095208481704

SARIMA(0, 1, 0)x(1, 0, 1, 12)12 - AIC:926.5281532938114

SARIMA(0, 1, 0)x(1, 1, 0, 12)12 - AIC:808.6771142223469

SARIMA(0, 1, 0)x(1, 1, 1, 12)12 - AIC:810.5955007326576

SARIMA(0, 1, 1)x(0, 0, 0, 12)12 - AIC:1127.0198500356564

SARIMA(0, 1, 1)x(0, 0, 1, 12)12 - AIC:1046.5034335213322

SARIMA(0, 1, 1)x(0, 1, 0, 12)12 - AIC:803.6498821399686

SARIMA(0, 1, 1)x(0, 1, 1, 12)12 - AIC:805.4084779408394

SARIMA(0, 1, 1)x(1, 0, 0, 12)12 - AIC:923.5578354629084

SARIMA(0, 1, 1)x(1, 0, 1, 12)12 - AIC:925.3345143968677

SARIMA(0, 1, 1)x(1, 1, 0, 12)12 - AIC:805.3394040910772

SARIMA(0, 1, 1)x(1, 1, 1, 12)12 - AIC:806.4195017018053

SARIMA(1, 0, 0)x(0, 0, 0, 12)12 - AIC:1154.3467824503816

SARIMA(1, 0, 0)x(0, 0, 1, 12)12 - AIC:1064.745248052037

SARIMA(1, 0, 0)x(0, 1, 0, 12)12 - AIC:815.6122869236708

SARIMA(1, 0, 0)x(0, 1, 1, 12)12 - AIC:817.1433310735763

SARIMA(1, 0, 0)x(1, 0, 0, 12)12 - AIC:936.551897797727

SARIMA(1, 0, 0)x(1, 0, 1, 12)12 - AIC:938.1151857952209

SARIMA(1, 0, 0)x(1, 1, 0, 12)12 - AIC:817.0634306591082

SARIMA(1, 0, 0)x(1, 1, 1, 12)12 - AIC:816.7179267149751

SARIMA(1, 0, 1)x(0, 0, 0, 12)12 - AIC:1142.3956180517923

SARIMA(1, 0, 1)x(0, 0, 1, 12)12 - AIC:1060.6066110152572

SARIMA(1, 0, 1)x(0, 1, 0, 12)12 - AIC:812.7466412386547

SARIMA(1, 0, 1)x(0, 1, 1, 12)12 - AIC:814.5974836335199

SARIMA(1, 0, 1)x(1, 0, 0, 12)12 - AIC:935.7192824124853

SARIMA(1, 0, 1)x(1, 0, 1, 12)12 - AIC:937.547888706627

SARIMA(1, 0, 1)x(1, 1, 0, 12)12 - AIC:814.5514800157281

SARIMA(1, 0, 1)x(1, 1, 1, 12)12 - AIC:811.7859832038592

SARIMA(1, 1, 0)x(0, 0, 0, 12)12 - AIC:1130.6563244373124

SARIMA(1, 1, 0)x(0, 0, 1, 12)12 - AIC:1046.244884079866

SARIMA(1, 1, 0)x(0, 1, 0, 12)12 - AIC:803.2812826905046

SARIMA(1, 1, 0)x(0, 1, 1, 12)12 - AIC:805.0435224122654

SARIMA(1, 1, 0)x(1, 0, 0, 12)12 - AIC:923.3218239541696

SARIMA(1, 1, 0)x(1, 0, 1, 12)12 - AIC:925.1067649481432

SARIMA(1, 1, 0)x(1, 1, 0, 12)12 - AIC:804.9733107098839

SARIMA(1, 1, 0)x(1, 1, 1, 12)12 - AIC:806.0045333987778

SARIMA(1, 1, 1)x(0, 0, 0, 12)12 - AIC:1125.4285934222276

SARIMA(1, 1, 1)x(0, 0, 1, 12)12 - AIC:1047.4625749926154

SARIMA(1, 1, 1)x(0, 1, 0, 12)12 - AIC:804.8133348303301

SARIMA(1, 1, 1)x(0, 1, 1, 12)12 - AIC:806.570146110246

SARIMA(1, 1, 1)x(1, 0, 0, 12)12 - AIC:924.5502292967215

SARIMA(1, 1, 1)x(1, 0, 1, 12)12 - AIC:926.3624606145682

SARIMA(1, 1, 1)x(1, 1, 0, 12)12 - AIC:806.5003437997995

SARIMA(1, 1, 1)x(1, 1, 1, 12)12 - AIC:807.6603076307986

SARIMA(1, 1, 0)x(0, 1, 0, 12)12 - AIC:803.2812826905046

In [34]:

model **=** SARIMAX(train, order**=**best\_order, seasonal\_order**=**best\_seasonal\_order)

sarima\_final\_model **=** model**.**fit(disp**=**0)

In [35]:

y\_pred\_test **=** sarima\_final\_model**.**get\_forecast(steps**=**24)

In [36]:

y\_pred **=** y\_pred\_test**.**predicted\_mean

In [37]:

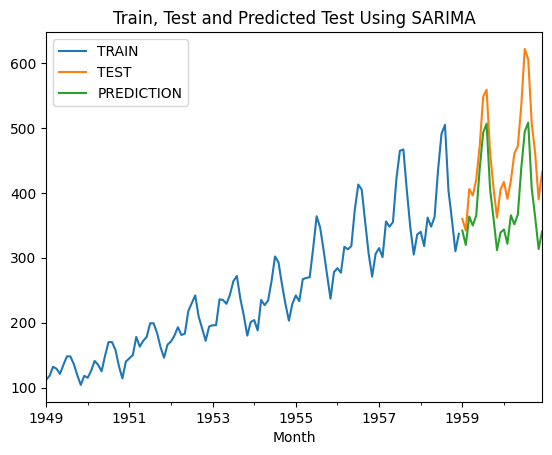
mean\_absolute\_error(test, y\_pred)

Out[37]:

68.5772654544699

In [38]:

plot\_prediction(pd**.**Series(y\_pred, index**=**test**.**index), "SARIMA")



It was also able to capture seasonality in the time series.

**SARIMA Optimization - MAE**

In [39]:

p **=** d **=** q **=** range(0, 2)

pdq **=** list(itertools**.**product(p, d, q))

seasonal\_pdq **=** [(x[0], x[1], x[2], 12) **for** x **in** list(itertools**.**product(p, d, q))]

In [40]:

**def** sarima\_optimizer\_mae(train, pdq, seasonal\_pdq):

best\_mae, best\_order, best\_seasonal\_order **=** float("inf"), float("inf"), **None**

**for** param **in** pdq:

**for** param\_seasonal **in** seasonal\_pdq:

**try**:

model **=** SARIMAX(train, order**=**param, seasonal\_order**=**param\_seasonal)

sarima\_model **=** model**.**fit(disp**=**0)

y\_pred\_test **=** sarima\_model**.**get\_forecast(steps**=**24)

y\_pred **=** y\_pred\_test**.**predicted\_mean

mae **=** mean\_absolute\_error(test, y\_pred)

*# mae = fit\_model\_sarima(train, val, param, param\_seasonal)*

**if** mae **<** best\_mae:

best\_mae, best\_order, best\_seasonal\_order **=** mae, param, param\_seasonal

print('SARIMA{}x{}12 - MAE:{}'**.**format(param, param\_seasonal, mae))

**except**:

**continue**

print('SARIMA{}x{}12 - MAE:{}'**.**format(best\_order, best\_seasonal\_order, best\_mae))

**return** best\_order, best\_seasonal\_order

In [41]:

best\_order, best\_seasonal\_order **=** sarima\_optimizer\_mae(train, pdq, seasonal\_pdq)

SARIMA(0, 0, 0)x(0, 0, 0, 12)12 - MAE:452.25

SARIMA(0, 0, 0)x(0, 0, 1, 12)12 - MAE:367.2095457540031

SARIMA(0, 0, 0)x(0, 1, 0, 12)12 - MAE:71.25

SARIMA(0, 0, 0)x(0, 1, 1, 12)12 - MAE:72.62972633135242

SARIMA(0, 0, 0)x(1, 0, 0, 12)12 - MAE:74.93608146912112

SARIMA(0, 0, 0)x(1, 0, 1, 12)12 - MAE:76.79705064327548

SARIMA(0, 0, 0)x(1, 1, 0, 12)12 - MAE:58.599738774342704

SARIMA(0, 0, 0)x(1, 1, 1, 12)12 - MAE:30.63273290975219

SARIMA(0, 0, 1)x(0, 0, 0, 12)12 - MAE:442.6732363260401

SARIMA(0, 0, 1)x(0, 0, 1, 12)12 - MAE:356.28978572187

SARIMA(0, 0, 1)x(0, 1, 0, 12)12 - MAE:70.4925284843363

SARIMA(0, 0, 1)x(0, 1, 1, 12)12 - MAE:71.32051669748289

SARIMA(0, 0, 1)x(1, 0, 0, 12)12 - MAE:75.4845073678041

SARIMA(0, 0, 1)x(1, 0, 1, 12)12 - MAE:76.43277472516246

SARIMA(0, 0, 1)x(1, 1, 0, 12)12 - MAE:61.0138854963242

SARIMA(0, 0, 1)x(1, 1, 1, 12)12 - MAE:31.821837528011624

SARIMA(0, 1, 0)x(0, 0, 0, 12)12 - MAE:115.25

SARIMA(0, 1, 0)x(0, 0, 1, 12)12 - MAE:119.36368412754128

SARIMA(0, 1, 0)x(0, 1, 0, 12)12 - MAE:69.74999999999982

SARIMA(0, 1, 0)x(0, 1, 1, 12)12 - MAE:66.94114692659889

SARIMA(0, 1, 0)x(1, 0, 0, 12)12 - MAE:73.24103227043334

SARIMA(0, 1, 0)x(1, 0, 1, 12)12 - MAE:70.5976716083623

SARIMA(0, 1, 0)x(1, 1, 0, 12)12 - MAE:66.58399646984448

SARIMA(0, 1, 0)x(1, 1, 1, 12)12 - MAE:66.48207360759267

SARIMA(0, 1, 1)x(0, 0, 0, 12)12 - MAE:99.11579006627255

SARIMA(0, 1, 1)x(0, 0, 1, 12)12 - MAE:116.07824672963376

SARIMA(0, 1, 1)x(0, 1, 0, 12)12 - MAE:67.81726173070768

SARIMA(0, 1, 1)x(0, 1, 1, 12)12 - MAE:66.40248616560132

SARIMA(0, 1, 1)x(1, 0, 0, 12)12 - MAE:71.54034585027608

SARIMA(0, 1, 1)x(1, 0, 1, 12)12 - MAE:70.00583318530141

SARIMA(0, 1, 1)x(1, 1, 0, 12)12 - MAE:65.99354814573655

SARIMA(0, 1, 1)x(1, 1, 1, 12)12 - MAE:66.13779086138842

SARIMA(1, 0, 0)x(0, 0, 0, 12)12 - MAE:140.44698908336036

SARIMA(1, 0, 0)x(0, 0, 1, 12)12 - MAE:141.32884915003453

SARIMA(1, 0, 0)x(0, 1, 0, 12)12 - MAE:70.39282960557512

SARIMA(1, 0, 0)x(0, 1, 1, 12)12 - MAE:69.50233161829148

SARIMA(1, 0, 0)x(1, 0, 0, 12)12 - MAE:81.06878462353522

SARIMA(1, 0, 0)x(1, 0, 1, 12)12 - MAE:79.02604486322248

SARIMA(1, 0, 0)x(1, 1, 0, 12)12 - MAE:69.2331611178334

SARIMA(1, 0, 0)x(1, 1, 1, 12)12 - MAE:57.791318899439936

SARIMA(1, 0, 1)x(0, 0, 0, 12)12 - MAE:139.15809373196245

SARIMA(1, 0, 1)x(0, 0, 1, 12)12 - MAE:143.35914388851995

SARIMA(1, 0, 1)x(0, 1, 0, 12)12 - MAE:69.02722305834956

SARIMA(1, 0, 1)x(0, 1, 1, 12)12 - MAE:68.36463043894311

SARIMA(1, 0, 1)x(1, 0, 0, 12)12 - MAE:77.90259429102936

SARIMA(1, 0, 1)x(1, 0, 1, 12)12 - MAE:76.77383192259231

SARIMA(1, 0, 1)x(1, 1, 0, 12)12 - MAE:68.13029386159323

SARIMA(1, 0, 1)x(1, 1, 1, 12)12 - MAE:57.748988727571714

SARIMA(1, 1, 0)x(0, 0, 0, 12)12 - MAE:105.03411924674504

SARIMA(1, 1, 0)x(0, 0, 1, 12)12 - MAE:114.85570882872896

SARIMA(1, 1, 0)x(0, 1, 0, 12)12 - MAE:68.5772654544699

SARIMA(1, 1, 0)x(0, 1, 1, 12)12 - MAE:67.15217192999283

SARIMA(1, 1, 0)x(1, 0, 0, 12)12 - MAE:71.98109072088141

SARIMA(1, 1, 0)x(1, 0, 1, 12)12 - MAE:70.45326032616391

SARIMA(1, 1, 0)x(1, 1, 0, 12)12 - MAE:66.73095833060371

SARIMA(1, 1, 0)x(1, 1, 1, 12)12 - MAE:67.05903505000775

SARIMA(1, 1, 1)x(0, 0, 0, 12)12 - MAE:93.90498940778257

SARIMA(1, 1, 1)x(0, 0, 1, 12)12 - MAE:113.40792528004285

SARIMA(1, 1, 1)x(0, 1, 0, 12)12 - MAE:69.8028255074954

SARIMA(1, 1, 1)x(0, 1, 1, 12)12 - MAE:68.34977165501536

SARIMA(1, 1, 1)x(1, 0, 0, 12)12 - MAE:73.46701320453475

SARIMA(1, 1, 1)x(1, 0, 1, 12)12 - MAE:71.98883358701319

SARIMA(1, 1, 1)x(1, 1, 0, 12)12 - MAE:67.92901703437411

SARIMA(1, 1, 1)x(1, 1, 1, 12)12 - MAE:67.58763872499561

SARIMA(0, 0, 0)x(1, 1, 1, 12)12 - MAE:30.63273290975219

In [42]:

model **=** SARIMAX(train, order**=**best\_order, seasonal\_order**=**best\_seasonal\_order)

sarima\_final\_model **=** model**.**fit(disp**=**0)

In [43]:

y\_pred\_test **=** sarima\_final\_model**.**get\_forecast(steps**=**24)

In [44]:

y\_pred **=** y\_pred\_test**.**predicted\_mean

In [45]:

mean\_absolute\_error(test, y\_pred)

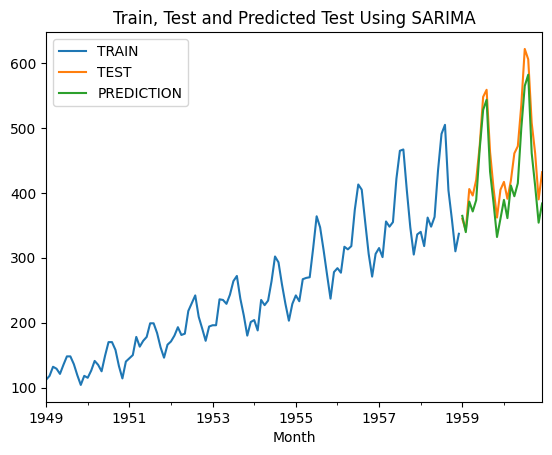
Out[45]:

30.63273290975219

It gave better results than AIC.

In [46]:

plot\_prediction(pd**.**Series(y\_pred, index**=**test**.**index), "SARIMA")



**Final Model**

In [47]:

pip install nbformat **--**upgrade

Requirement already satisfied: nbformat in /Users/jigishap/.pyenv/versions/3.10.6/envs/Airline/lib/python3.10/site-packages (5.9.2)

Requirement already satisfied: fastjsonschema in /Users/jigishap/.pyenv/versions/3.10.6/envs/Airline/lib/python3.10/site-packages (from nbformat) (2.19.1)

Requirement already satisfied: jsonschema>=2.6 in /Users/jigishap/.pyenv/versions/3.10.6/envs/Airline/lib/python3.10/site-packages (from nbformat) (4.20.0)

Requirement already satisfied: jupyter-core in /Users/jigishap/.pyenv/versions/3.10.6/envs/Airline/lib/python3.10/site-packages (from nbformat) (5.6.1)

Requirement already satisfied: traitlets>=5.1 in /Users/jigishap/.pyenv/versions/3.10.6/envs/Airline/lib/python3.10/site-packages (from nbformat) (5.14.1)

Requirement already satisfied: attrs>=22.2.0 in /Users/jigishap/.pyenv/versions/3.10.6/envs/Airline/lib/python3.10/site-packages (from jsonschema>=2.6->nbformat) (23.2.0)

Requirement already satisfied: jsonschema-specifications>=2023.03.6 in /Users/jigishap/.pyenv/versions/3.10.6/envs/Airline/lib/python3.10/site-packages (from jsonschema>=2.6->nbformat) (2023.12.1)

Requirement already satisfied: referencing>=0.28.4 in /Users/jigishap/.pyenv/versions/3.10.6/envs/Airline/lib/python3.10/site-packages (from jsonschema>=2.6->nbformat) (0.32.0)

Requirement already satisfied: rpds-py>=0.7.1 in /Users/jigishap/.pyenv/versions/3.10.6/envs/Airline/lib/python3.10/site-packages (from jsonschema>=2.6->nbformat) (0.16.2)

Requirement already satisfied: platformdirs>=2.5 in /Users/jigishap/.pyenv/versions/3.10.6/envs/Airline/lib/python3.10/site-packages (from jupyter-core->nbformat) (4.1.0)

Note: you may need to restart the kernel to use updated packages.

In [48]:

data **=** {

"Models": ["SES", "DES", "TES", "ARIMA", "SARIMA"],

"MAE": [82.528, 54.1036, 11.9947, 64.0122, 30.6261]

}

df\_models **=** pd**.**DataFrame(data)

fig **=** px**.**bar(df\_models, x**=**'Models', y**=**'MAE', color**=**'Models',

labels**=**{'MAE': 'MAE'},

text**=**'MAE', title**=**"MAE Values for Different Models")

*#fig.update\_traces(texttemplate='%{text:.4f}', textposition='outside')*

*#fig.update\_layout(uniformtext\_minsize=12, uniformtext\_mode='hide')*

fig**.**show()

In [49]:

data **=** {

"Models": ["SES", "DES", "TES", "ARIMA", "SARIMA"],

"MAE": [82.528, 54.1036, 11.9947, 64.0122, 30.6261]

}

df\_models **=** pd**.**DataFrame(data)

plt**.**bar(df\_models['Models'], df\_models['MAE'], color**=**'skyblue')

plt**.**xlabel('Models')

plt**.**ylabel('MAE')

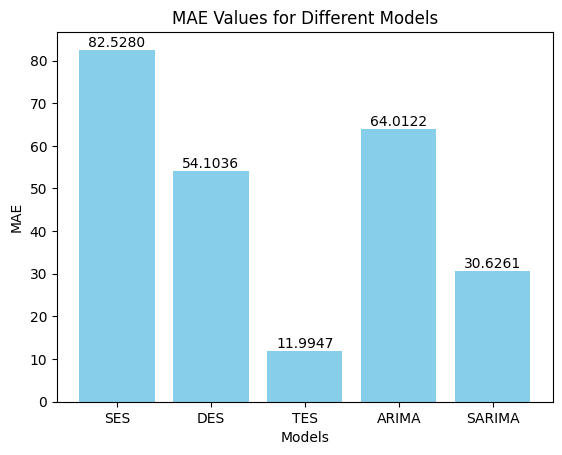
plt**.**title('MAE Values for Different Models')

*# Add data labels*

**for** i, value **in** enumerate(df\_models['MAE']):

plt**.**text(i, value, f'{value:.4f}', ha**=**'center', va**=**'bottom')

plt**.**show()



When we evaluated the MAE values, we achieved the best result with TES. We will build our final model with TES and look at the prediction results.

In [50]:

tes\_model\_final **=** ExponentialSmoothing(df, trend**=**"add", seasonal**=**"add", seasonal\_periods**=**12)**.**\

fit(smoothing\_level**=**best\_alpha, smoothing\_slope**=**best\_beta, smoothing\_seasonal**=**best\_gamma)

In [51]:

tes\_model\_final**.**forecast(6)

Out[51]:

1961-01-01 450.286319

1961-02-01 429.302004

1961-03-01 475.721697

1961-04-01 502.017546

1961-05-01 516.825551

1961-06-01 579.037163

Freq: MS, dtype: float64

The last number of passengers in our dataset was in December 1960. We have now been able to estimate passenger numbers for the first 6 months of 1961. An increasing trend has been observed.

**Summary**