Analisi delle serie temporali

Homework n.ro 2: Information Systems and Business Intelligence

Federica Del Vecchio (M63001587)
Claudia Antonella Esposito (M63001625)
Tommaso Di Lillo (M63001642)

Obiettivo

Analisi delle feature *Numero di insetti* e *Nuove catture (per evento)*, utilizzando oltre ai dati storici di queste variabili anche dati meteorologici

Tipo di problema:

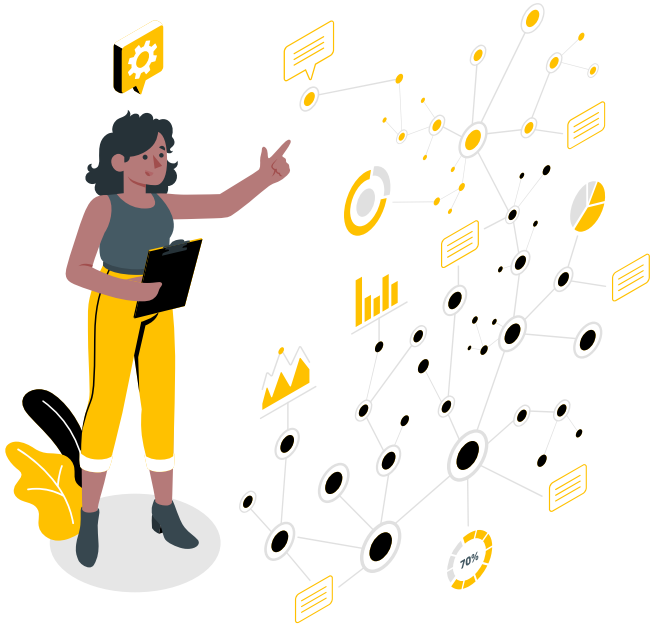
Regressione → Predizione dei valori degli attributi numerici: *Numero di insetti* e *Nuove catture (per evento)*

Strumenti:

- Google Drive
- Microsoft Excel
- Google Colab
- Librerie Python (ad esempio: Scikit-learn, Pandas, TensorFlow, Seaborn, Streamlit, ecc.)
- Ngrok

Pre-processing

01



Sono disponibili quattro file: due contengono dati meteorologici storici, mentre gli altri due riportano informazioni sulle catture degli insetti.

I dati meteo sono registrati su base oraria, mentre le catture sono documentate su base giornaliera.

Dati meteo storici Cicalino (1)

	DateTime	Media Temperatura	Media Umidità
0	05.07.2024 15:00:00	31.65	37.08
1	05.07.2024 16:00:00	31.45	33.55
2	05.07.2024 17:00:00	31.66	34.98
3	05.07.2024 18:00:00	30.08	42.39
4	05.07.2024 19:00:00	28.78	47.54
...	...	...	...
1163	23.08.2024 02:00:00	19.57	87.25
1164	23.08.2024 03:00:00	19.92	86.89
1165	23.08.2024 04:00:00	19.57	88.57
1166	23.08.2024 05:00:00	18.58	92.14
1167	23.08.2024 06:00:00	18.36	93.30

Grafico delle catture Cicalino (1)

	DateTime	Numero di insetti	Nuove catture (per evento)	Recensito	Evento
0	06.07.2024 06:01:00	0.0	0.0	Si	NaN
1	07.07.2024 06:04:00	0.0	0.0	Si	NaN
2	08.07.2024 06:03:00	0.0	0.0	Si	NaN
3	09.07.2024 06:05:00	0.0	0.0	Si	NaN
4	09.07.2024 06:13:59	NaN	NaN	Si	Cleaning
5	10.07.2024 06:03:00	0.0	0.0	Si	NaN
6	11.07.2024 06:01:00	0.0	0.0	Si	NaN
7	12.07.2024 06:05:00	0.0	0.0	Si	NaN
8	13.07.2024 06:02:00	0.0	0.0	Si	NaN
9	14.07.2024 06:01:00	1.0	1.0	Si	NaN
10	15.07.2024 06:04:00	1.0	0.0	Si	NaN
11	16.07.2024 06:03:00	2.0	1.0	Si	NaN
12	16.07.2024 06:12:59	NaN	NaN	Si	Cleaning

...

La colonna *Date* è stata impostata come indice per tutti i DataFrame Pandas.

```
# Conversione della colonna "DateTime" in formato datetime per ciascun DataFrame
df1["DateTime"] = pd.to_datetime(df1["DateTime"], format="%d.%m.%Y %H:%M:%S", errors='coerce')
df2['DateTime'] = pd.to_datetime(df2['DateTime'], format="%d.%m.%Y %H:%M:%S", errors='coerce')
df3['DateTime'] = pd.to_datetime(df3['DateTime'], format="%d.%m.%Y %H:%M:%S", errors='coerce')
df4['DateTime'] = pd.to_datetime(df4['DateTime'], format="%d.%m.%Y %H:%M:%S", errors='coerce')

# Impostazione della colonna "DateTime" come indice per ciascun DataFrame
df1.index = df1.DateTime
del df1["DateTime"]

df2.index = df2.DateTime
del df2["DateTime"]

df3.index = df3.DateTime
del df3["DateTime"]

df4.index = df4.DateTime
del df4["DateTime"]
```

DateTime	Media Temperatura	Media Umidità
2024-07-05 15:00:00	31.65	37.08
2024-07-05 16:00:00	31.45	33.55
2024-07-05 17:00:00	31.66	34.98
2024-07-05 18:00:00	30.08	42.39
2024-07-05 19:00:00	28.78	47.54
...	...	...

È stato eseguito un resample dell'indice, raggruppando i dati metereologici su base giornaliera.

Successivamente, i due DataFrame, aventi indici parzialmente sovrapposti, sono stati concatenati. Per i giorni condivisi, i valori di temperatura e umidità sono stati mediati, sostituendo così le righe originali.

```
# Raggruppa per giorno e calcola la media
df1_daily = df1.resample('D').mean()  # 'D' indica raggruppamento giornaliero
# Raggruppa per giorno e calcola la media
df2_daily = df2.resample('D').mean()
```

```
df_merged = pd.concat([df1_daily, df2_daily])
df_merged  = df_merged.groupby(df_merged.index).mean()
```

DateTime	Media Temperatura	Media Umidità
2024-07-05	25.777778	56.674444
2024-07-06	22.342500	72.247500
2024-07-07	23.517917	76.728750
2024-07-08	25.669167	69.143750
2024-07-09	25.870000	53.647083
2024-07-10	26.410000	58.943750
2024-07-11	26.214511	55.223995
2024-07-12	26.363750	57.509583
2024-07-13	25.699792	68.389583
2024-07-14	26.193750	68.048125
2024-07-15	27.143542	56.963750

...

Per i DataFrame contenenti informazioni sul fenomeno degli insetti, la prima operazione di pre-processamento consiste nella normalizzazione degli indici, rimuovendo l'orario e mantenendo solo la data, che rappresenta l'informazione di interesse.

Output

DateTime	Numero di insetti	Nuove catture (per evento)
2024-07-05	0.0	0.0
2024-07-06	0.0	0.0
2024-07-07	0.0	0.0
2024-07-08	0.0	0.0
2024-07-09	0.0	0.0
2024-07-10	0.0	0.0
2024-07-11	0.0	0.0
2024-07-12	0.0	0.0
2024-07-13	0.0	0.0
2024-07-14	1.0	1.0
2024-07-15	1.0	0.0
2024-07-16	2.0	1.0
2024-07-17	1.0	1.0

...

```
df3.index = df3.index.normalize()
df4.index = df4.index.normalize()
```

```
df3 = df3.drop(columns=['Recensito', 'Evento'])
df4 = df4.drop(columns=['Recensito', 'Evento'])
```

```
df3 = df3.fillna(0)
df4 = df4.fillna(0)
```

```
df_merged2 = pd.concat([df3, df4])
df_merged2  = df_merged2.groupby(df_merged2.index).sum()
```

I valori mancanti («NaN»), derivanti da eventi di «Cleaning», sono stati considerati non rilevanti. Di conseguenza, sono state eliminate le colonne *Recensito* ed *Evento*, insieme alle righe contenenti «NaN».

visualize_dataframe_info: funzione di ispezione di un DataFrame

```python
def visualize_dataframe_info(df, title):
    # Genera il contenuto da visualizzare
    print(f"\n{'-'*56}\n{title}\n{'-'*56}")
    # Verifica se l'indice è una data e, in caso positivo, mostra l'intervallo di date
    if pd.api.types.is_datetime64_any_dtype(df.index):
        print(f"L'indice è una data, intervallo: {df.index.min().strftime('%d-%m-%Y')} a {df.index.max().strftime('%d-%m-%Y')}")
    else:
        print("L'indice non è una data")
    # Numero di righe e colonne
    print(f"\nNumero di righe: {df.shape[0]}")
    print(f"Numero di colonne: {df.shape[1]}")
    # Nome e tipo delle colonne
    print("\nColonne e tipi:")
    for col, dtype in df.dtypes.items():
        print(f"{col}: {dtype}")
    # Verifica valori nulli
    print("\nValori nulli per colonna:")
    for col in df.columns:
        print(f"{col}: {df.isnull().sum()[col]}")

    # Statistiche delle colonne numeriche (Max, Min, Media)
    print("\nStatistiche (Max, Min, Media) per ogni colonna numerica:")
    desc = df.describe().loc[['max', 'min']].T
    mean_values = df.mean().round(3)  # Calcola la media e arrotonda a 3 decimali

    for col in desc.index:
        max_val = round(desc.loc[col, 'max'], 3)
        min_val = round(desc.loc[col, 'min'], 3)
        print(f"{col}: max={max_val}, min={min_val}")
    # Numero di valori distinti per colonna
    print("\nNumero di valori distinti per colonna:")
    for col in df.columns:
        print(f"{col}: {df[col].nunique()} distinti")
    # Per le colonne con meno di 10 valori distinti, visualizza i valori unici e la loro frequenza
    for col in df.columns:
        if df[col].nunique() < 10:
            print(f"\nColonna: {col}")
            value_counts = df[col].value_counts()
            for val, count in value_counts.items():
                print(f"Valore: {val}, Frequenza: {count}")
```

Output

```
------------------------------------------------------------
Dati Meteorologici (df_merged)
------------------------------------------------------------
L'indice è una data, intervallo: 05-07-2024 a 23-08-2024

Numero di righe: 50
Numero di colonne: 2

Colonne e tipi:
Media Temperatura: float64
Media Umidità: float64

Valori nulli per colonna:
Media Temperatura: 0
Media Umidità: 0

Statistiche (Max, Min, Media) per ogni colonna numerica:
Media Temperatura: max=30.594, min=19.676
Media Umidità: max=88.743, min=45.544

Numero di valori distinti per colonna:
Media Temperatura: 50 distinti
Media Umidità: 50 distinti
```

```
------------------------------------------------------------
Catture Insetti (df_merged2)
------------------------------------------------------------
L'indice è una data, intervallo: 05-07-2024 a 23-08-2024

Numero di righe: 50
Numero di colonne: 2

Colonne e tipi:
Numero di insetti: float64
Nuove catture (per evento): float64

Valori nulli per colonna:
Numero di insetti: 0
Nuove catture (per evento): 0

Statistiche (Max, Min, Media) per ogni colonna numerica:
Numero di insetti: max=4.0, min=0.0
Nuove catture (per evento): max=2.0, min=0.0

Numero di valori distinti per colonna:
Numero di insetti: 5 distinti
Nuove catture (per evento): 3 distinti

Colonna: Numero di insetti
Valore: 0.0, Frequenza: 30
Valore: 1.0, Frequenza: 10
Valore: 2.0, Frequenza: 6
Valore: 3.0, Frequenza: 3
Valore: 4.0, Frequenza: 1

Colonna: Nuove catture (per evento)
Valore: 0.0, Frequenza: 38
Valore: 1.0, Frequenza: 10
Valore: 2.0, Frequenza: 2
```

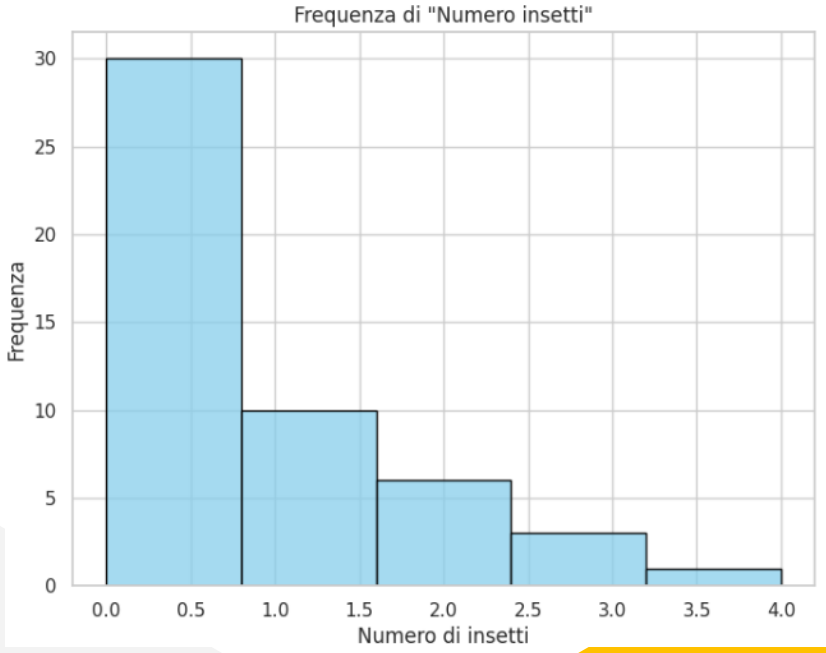
Infine, i due DataFrame (*df_merged* e *df_merged2*) vengono concatenati, unendo le loro colonne in base all'indice di tipo *DateTime*. Il risultato di questa operazione è un nuovo DataFrame, denominato *merged_df*, che per ogni giorno contiene informazioni sia sulle catture di insetti che sulla temperatura e umidità nella città di Cicalino.

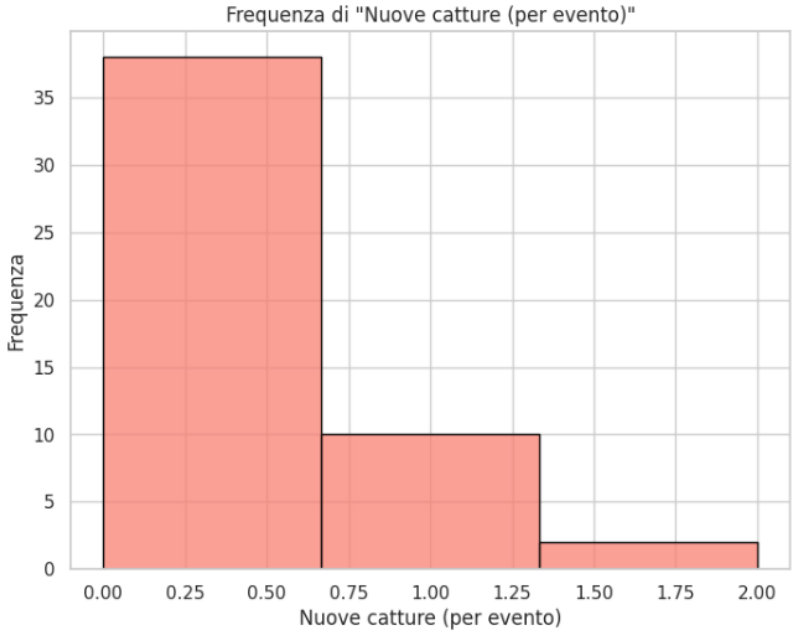
DateTime	Media Temperatura	Media Umidità	Numero di insetti	Nuove catture (per evento)
2024-07-05	25.777778	56.674444	0.0	0.0
2024-07-06	22.342500	72.247500	0.0	0.0
2024-07-07	23.517917	76.728750	0.0	0.0
2024-07-08	25.669167	69.143750	0.0	0.0
2024-07-09	25.870000	53.647083	0.0	0.0
2024-07-10	26.410000	58.943750	0.0	0.0
2024-07-11	26.214511	55.223995	0.0	0.0
2024-07-12	26.363750	57.509583	0.0	0.0
2024-07-13	25.699792	68.389583	0.0	0.0
2024-07-14	26.193750	68.048125	1.0	1.0
2024-07-15	27.143542	56.963750	1.0	0.0
2024-07-16	26.465417	56.074583	2.0	1.0
2024-07-17	27.547083	55.395000	1.0	1.0
2024-07-18	26.955833	52.736875	1.0	0.0
2024-07-19	27.801875	54.520000	2.0	1.0
2024-07-20	26.823542	63.676875	3.0	1.0

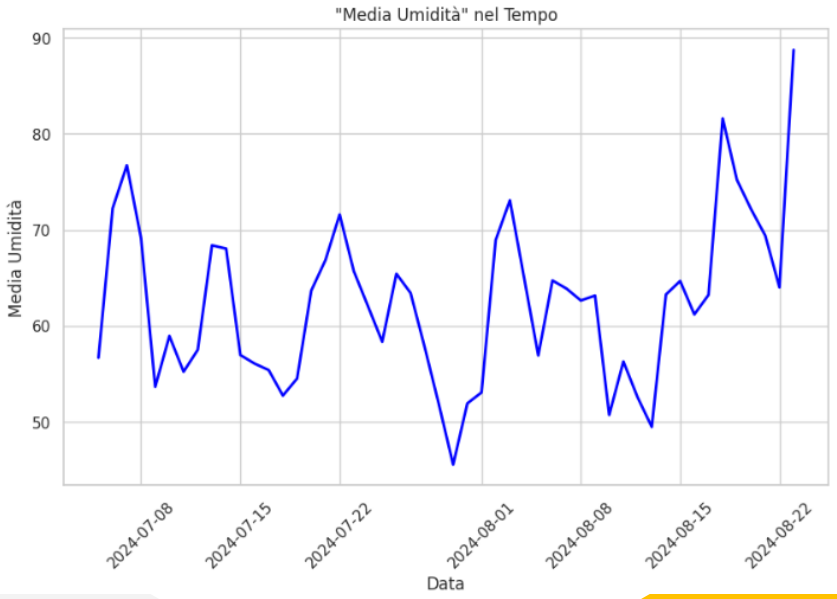
```
# Merge the dataframes
merged_df = pd.concat([df_merged, df_merged2], axis=1)
```

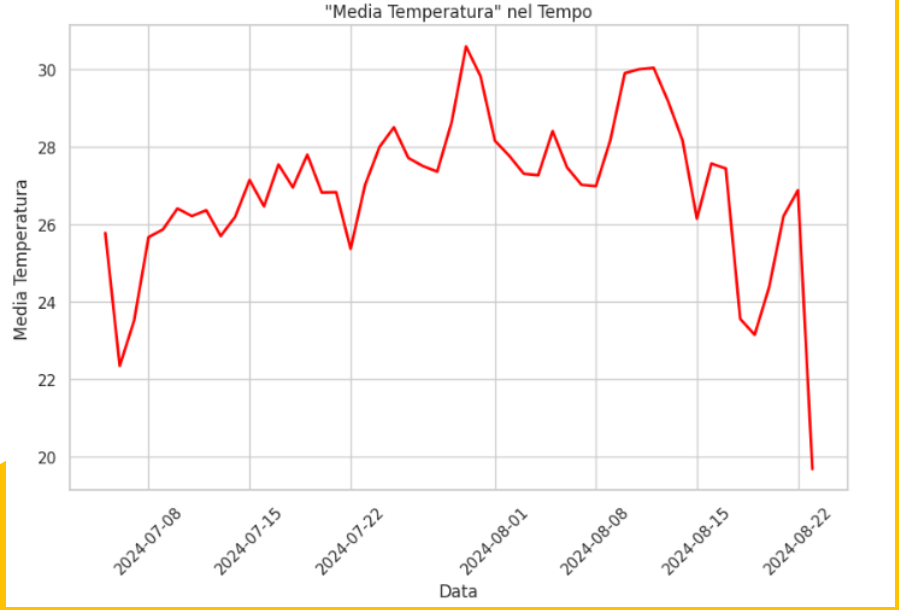
Visualizzazione dei dati

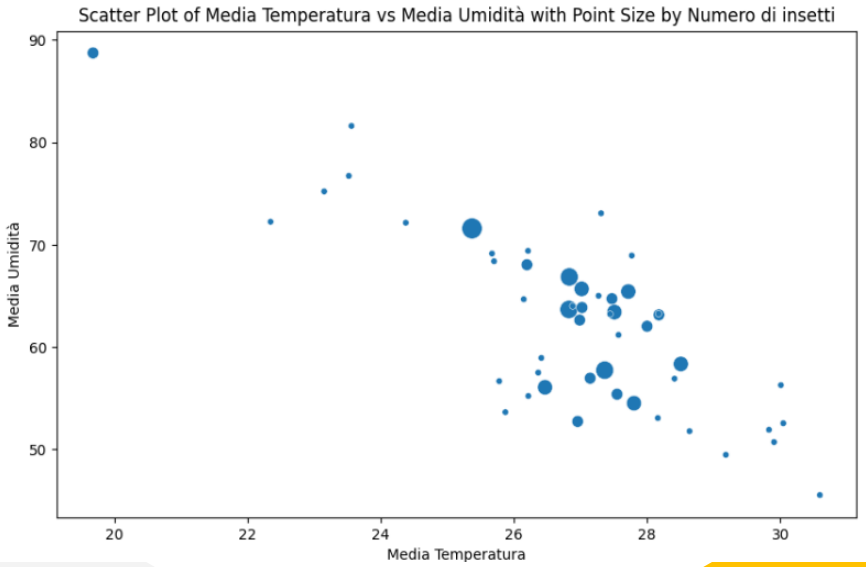
02

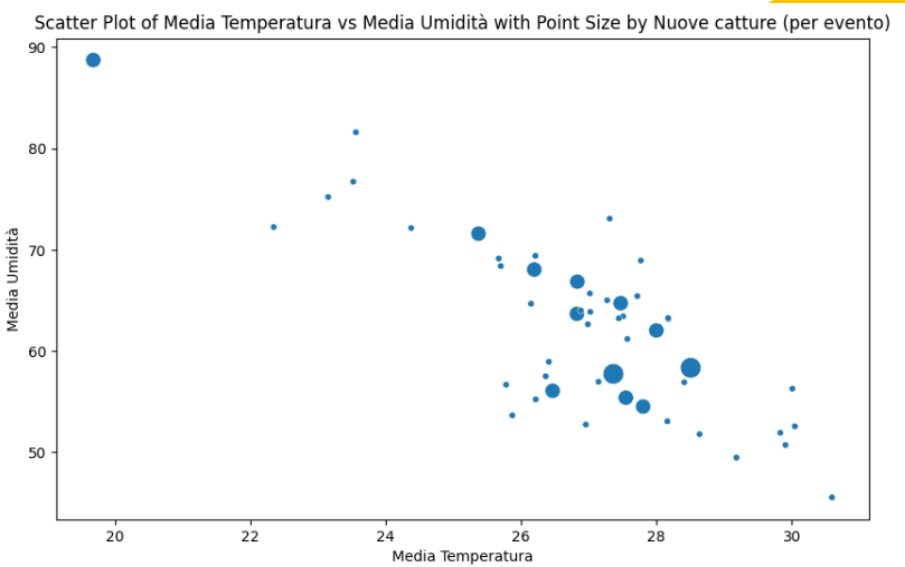

Frequenza di "Numero insetti"
Frequenza
Numero di insetti
0
5
10
15
20
25
30
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0


Frequenza di "Nuove catture (per evento)"
Frequenza
Nuove catture (per evento)
0
5
10
15
20
25
30
35
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00


"Media Umidità" nel Tempo
Media Umidità
90
80
70
60
50
2024-07-08
2024-07-15
2024-07-22
2024-08-01
2024-08-08
2024-08-15
2024-08-22
Data


"Media Temperatura" nel Tempo
Media Temperatura
30
28
26
24
22
20
2024-07-08
2024-07-15
2024-07-22
2024-08-01
2024-08-08
2024-08-15
2024-08-22
Data


Scatter Plot of Media Temperatura vs Media Umidità with Point Size by Numero di insetti
Media Umidità
Media Temperatura
90
80
70
60
50
20
22
24
26
28
30


Scatter Plot of Media Temperatura vs Media Umidità with Point Size by Nuove catture (per evento)
Media Umidità
Media Temperatura
90
80
70
60
50
20
22
24
26
28
30

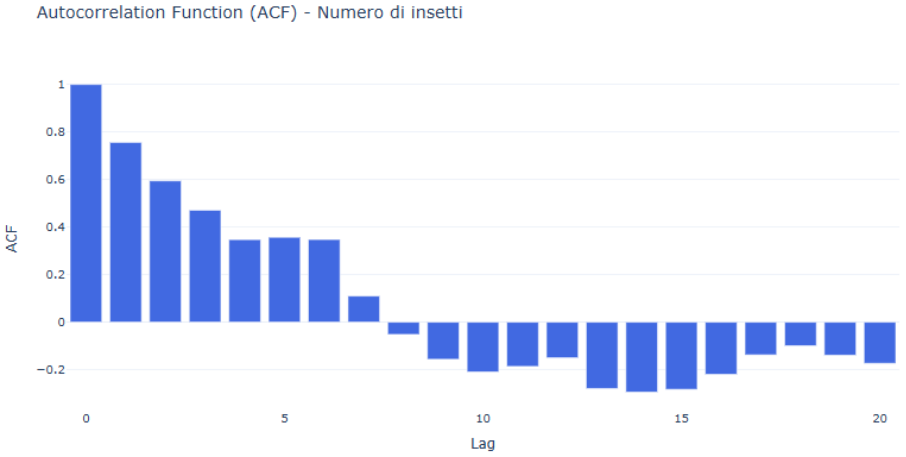
```python
def dataset_statistics(df):
    # Filtra solo le colonne numeriche dal DataFrame
    numeric_df = df.select_dtypes(include=[np.number])

    # Calcola statistiche descrittive (inclusi percentili) e trasponi per un formato leggibile
    stats = numeric_df.describe(percentiles=[.25, .5, .75]).T

    # Calcola metriche aggiuntive: asimmetria, curtosi, varianza e intervallo interquartile (IQR)
    stats['Skewness'] = numeric_df.skew()
    stats['Kurtosis'] = numeric_df.kurtosis()
    stats['Variance'] = numeric_df.var()
    stats['IQR'] = stats['75%'] - stats['25%']

    # Esegue il test di stazionarietà (ADF) per ogni colonna numerica
    adf_results = {}
    for column in numeric_df.columns:
        adf_test = adfuller(numeric_df[column].dropna())  # Elimina valori nulli per il test
        adf_results[column] = {
            'ADF Statistic': adf_test[0],  # Statistica del test ADF
            'p-value': adf_test[1],       # p-value per verificare la stazionarietà
            'Critical Values': adf_test[4]  # Valori critici del test
        }

    # Crea un DataFrame dai risultati del test ADF e conserva solo statistiche ADF e p-value
    adf_df = pd.DataFrame(adf_results).T
    adf_df = adf_df[['ADF Statistic', 'p-value']]

    # Combina le statistiche descrittive con i risultati ADF
    stats = pd.concat([stats, adf_df], axis=1)

    return stats
```

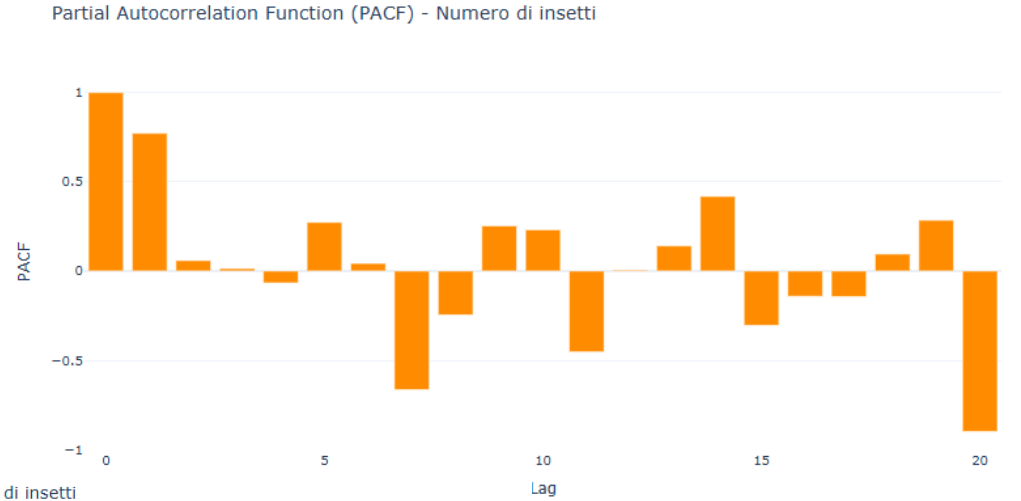
	count	mean	std	min	25%	50%	75%	max	Skewness	Kurtosis	Variance	IQR	ADF Statistic	p-value
Media Temperatura	50.0	26.900981	2.025520	19.675714	26.197969	27.082396	27.949219	30.594167	-1.098551	2.642206	4.102731	1.751250	-0.837516	0.807853
Media Umidità	50.0	62.405359	8.650729	45.544167	56.128021	63.189583	67.750208	88.742857	0.624642	0.655051	74.835120	11.622187	-3.029652	0.032217
Numero di insetti	50.0	0.700000	1.035098	0.000000	0.000000	0.000000	1.000000	4.000000	1.449138	1.367943	1.071429	1.000000	-2.52358	0.109842
Nuove catture (per evento)	50.0	0.280000	0.536048	0.000000	0.000000	0.000000	0.000000	2.000000	1.804559	2.513532	0.287347	0.000000	-1.991534	0.290297

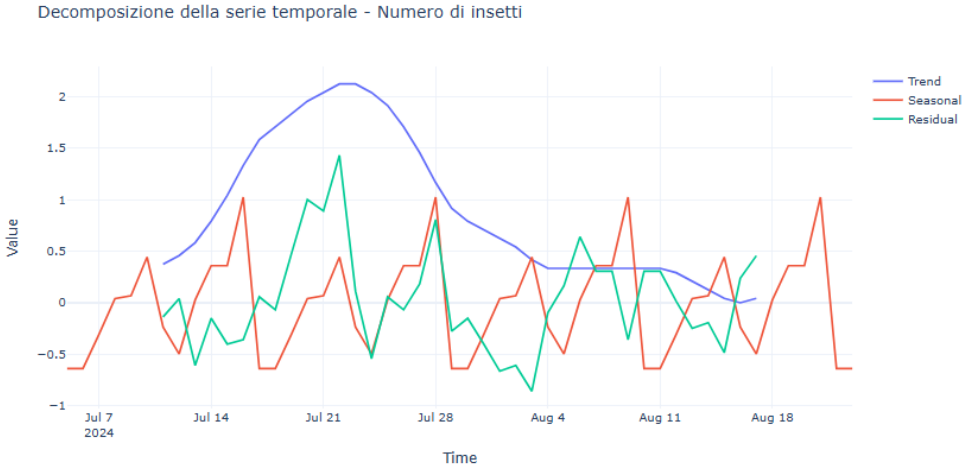
ACF & PACF plot

```python
# Calcolo e visualizzazione dell'Autocorrelation Function (ACF)
acf_values, confint = acf(time_series, nlags=lags, alpha=0.05)  # Calcolo dei valori ACF
fig_acf = go.Figure()
fig_acf.add_trace(go.Bar(x=np.arange(len(acf_values)), y=acf_values, name='ACF', marker_color='royalblue'))
fig_acf.update_layout(title=f"Autocorrelation Function (ACF) - {column}",
                      xaxis_title="Lag", yaxis_title="ACF", template="plotly_white")
fig_acf.show()

# Calcolo e visualizzazione della Partial Autocorrelation Function (PACF)
pacf_values, confint = pacf(time_series, nlags=lags, alpha=0.05)  # Calcolo dei valori PACF
fig_pacf = go.Figure()
fig_pacf.add_trace(go.Bar(x=np.arange(len(pacf_values)), y=pacf_values, name='PACF', marker_color='darkorange'))
fig_pacf.update_layout(title=f"Partial Autocorrelation Function (PACF) - {column}",
                       xaxis_title="Lag", yaxis_title="PACF", template="plotly_white")
fig_pacf.show()

# Decomposizione stagionale della serie temporale
result = seasonal_decompose(time_series.dropna(), model='additive', period=period)
fig_decomp = go.Figure()
fig_decomp.add_trace(go.Scatter(x=result.trend.index, y=result.trend, mode='lines', name='Trend'))
fig_decomp.add_trace(go.Scatter(x=result.seasonal.index, y=result.seasonal, mode='lines', name='Seasonal'))
fig_decomp.add_trace(go.Scatter(x=result.resid.index, y=result.resid, mode='lines', name='Residual'))
fig_decomp.update_layout(title=f"Decomposizione della serie temporale - {column}",
                         xaxis_title="Time", yaxis_title="Value", template="plotly_white")
fig_decomp.show()
```

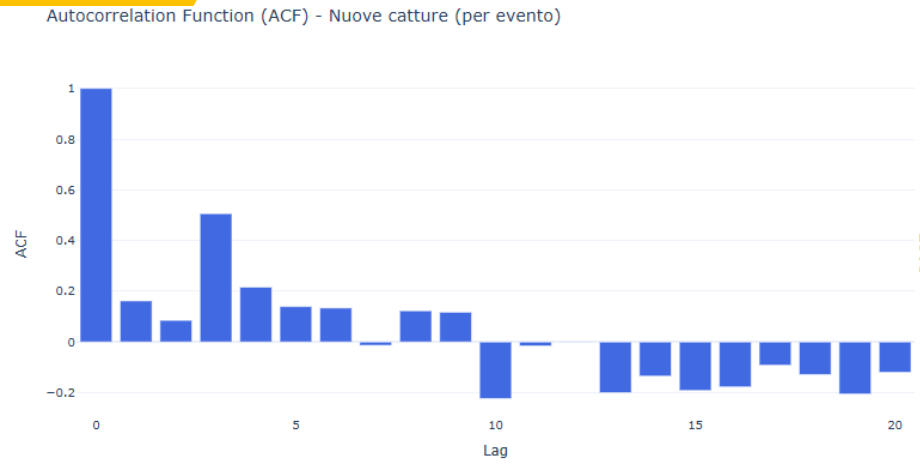
Numero di Insetti

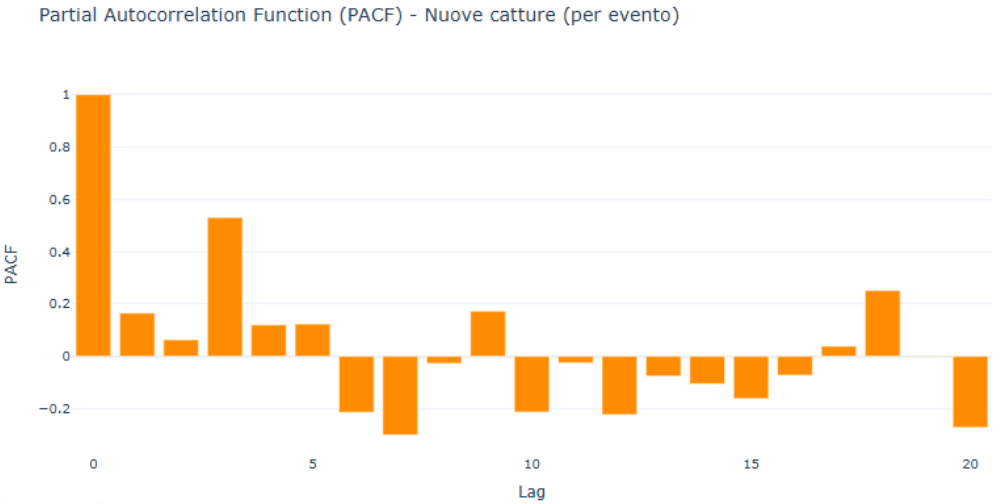


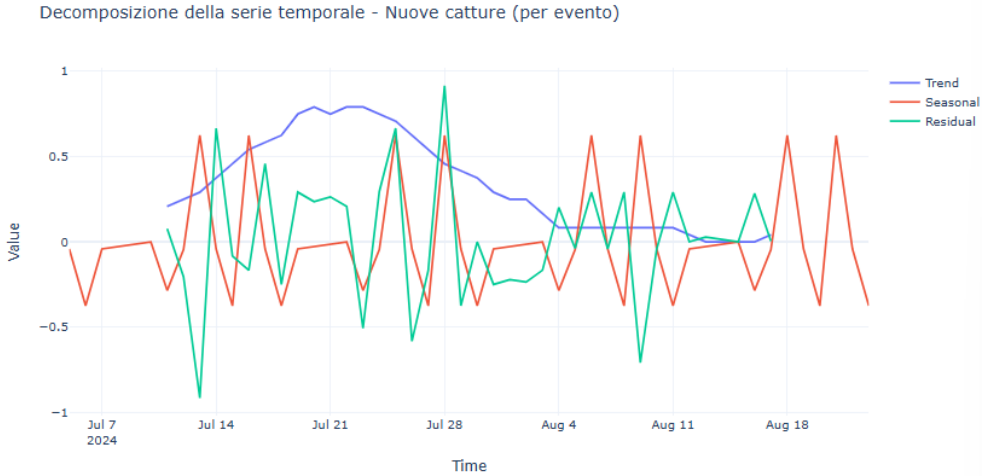




Nuove catture

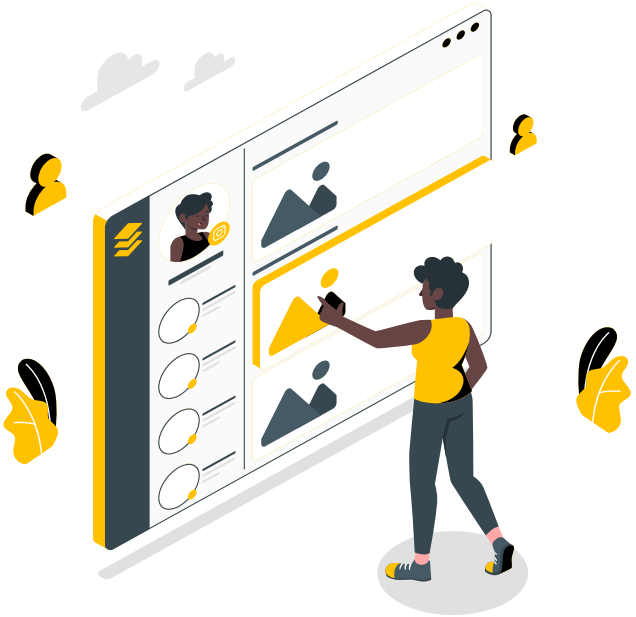






ARIMA

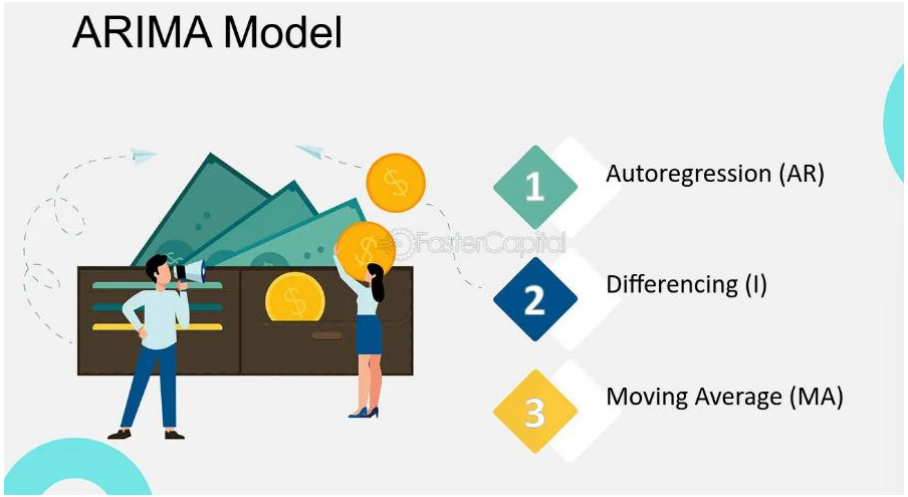
03



AutoRegressive Integrated Moving Average

Questo modello è composto da tre componenti principali:

- **AR** è la componente autoregressiva, ovvero la relazione tra la serie temporale e i suoi valori passati. Il parametro che definisce l'ordine di AR è **p**, cioè il numero di lag da considerare.
- **I** è la componente di differenziazione, che serve a rendere la serie stazionaria. Il parametro che definisce l'ordine di differenziazione è **d**.
- **MA** è la componente della media mobile, che rappresenta la relazione tra un valore della serie temporale e gli errori precedenti. Il parametro che definisce il numero di errori passati da considerare è **q**.



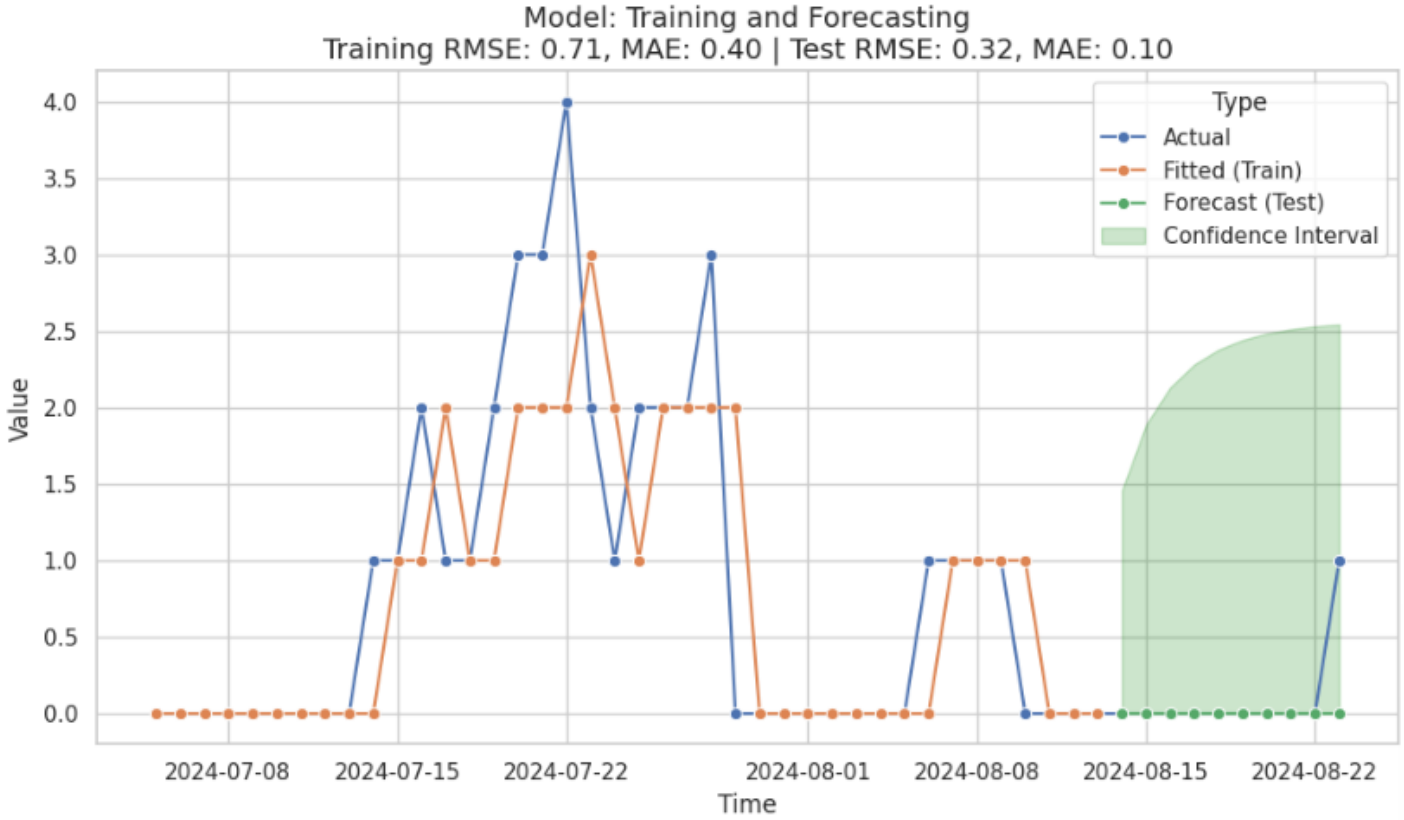
Il modello complessivo è rappresentato dalla notazione *ARIMA(p, d, q)*.

Invece, *AUTO-ARIMA* è una versione automatizzata del modello ARIMA che permette di individuare automaticamente i valori ottimali per i parametri p, d e q. AUTO-ARIMA esplora una serie di possibili combinazioni per questi parametri e utilizza criteri statistici come l'*AIC (Akaike Information Criterion)* e il *BIC (Bayesian Information Criterion)* per determinarne i valori migliori per la specifica serie temporale.

```
# Addestramento del modello ARIMAX con ricerca automatica dei parametri
model_1 = pm.auto_arima(
    y_train,
    exogenous=X_train,  # Variabili indipendenti
    seasonal=False,  # Modello non stagionale
    start_p=0, max_p=15,  # Range per il parametro p (autoregressione)
    start_q=0, max_q=15,  # Range per il parametro q (media mobile)
    max_d=5,  # Limite massimo per il differenziamento
    stepwise=False,  # Esplorazione completa dello spazio dei parametri
    trace=True  # Mostra i dettagli della ricerca
)
```

```
 ARIMA(0,0,0)(0,0,0)[0]             : AIC=141.189, Time=0.07 sec
 ARIMA(0,0,1)(0,0,0)[0]             : AIC=117.079, Time=0.03 sec
 ARIMA(0,0,2)(0,0,0)[0]             : AIC=110.972, Time=0.07 sec
 ARIMA(0,0,3)(0,0,0)[0]             : AIC=inf, Time=0.18 sec
 ARIMA(0,0,4)(0,0,0)[0]             : AIC=96.463, Time=0.14 sec
 ARIMA(0,0,5)(0,0,0)[0]             : AIC=98.270, Time=0.14 sec
 ARIMA(1,0,0)(0,0,0)[0]             : AIC=94.974, Time=0.04 sec
 ARIMA(1,0,1)(0,0,0)[0]             : AIC=96.567, Time=0.16 sec
 ARIMA(1,0,2)(0,0,0)[0]             : AIC=98.379, Time=0.52 sec
 ARIMA(1,0,3)(0,0,0)[0]             : AIC=100.376, Time=0.09 sec
 ARIMA(1,0,4)(0,0,0)[0]             : AIC=95.242, Time=0.15 sec
 ARIMA(2,0,0)(0,0,0)[0]             : AIC=96.617, Time=0.05 sec
 ARIMA(2,0,1)(0,0,0)[0]             : AIC=98.351, Time=0.07 sec
 ARIMA(2,0,2)(0,0,0)[0]             : AIC=100.356, Time=0.17 sec
 ARIMA(2,0,3)(0,0,0)[0]             : AIC=102.346, Time=0.35 sec
 ARIMA(3,0,0)(0,0,0)[0]             : AIC=98.493, Time=0.11 sec
 ARIMA(3,0,1)(0,0,0)[0]             : AIC=100.331, Time=0.10 sec
 ARIMA(3,0,2)(0,0,0)[0]             : AIC=inf, Time=3.09 sec
 ARIMA(4,0,0)(0,0,0)[0]             : AIC=100.484, Time=1.92 sec
 ARIMA(4,0,1)(0,0,0)[0]             : AIC=102.240, Time=0.26 sec
 ARIMA(5,0,0)(0,0,0)[0]             : AIC=99.315, Time=0.23 sec

Best model:  ARIMA(1,0,0)(0,0,0)[0]
Total fit time: 7.988 seconds
```

Numero di Insetti



model_1.summary()

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	40
Model:	SARIMAX(1, 0, 0)	**Log Likelihood**	-45.487
Date:	Fri, 13 Dec 2024	**AIC**	94.974
Time:	12:04:23	**BIC**	98.352
Sample:	07-05-2024 - 08-13-2024	**HQIC**	96.195
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.8235	0.052	15.861	0.000	0.722	0.925
sigma2	0.5533	0.094	5.863	0.000	0.368	0.738

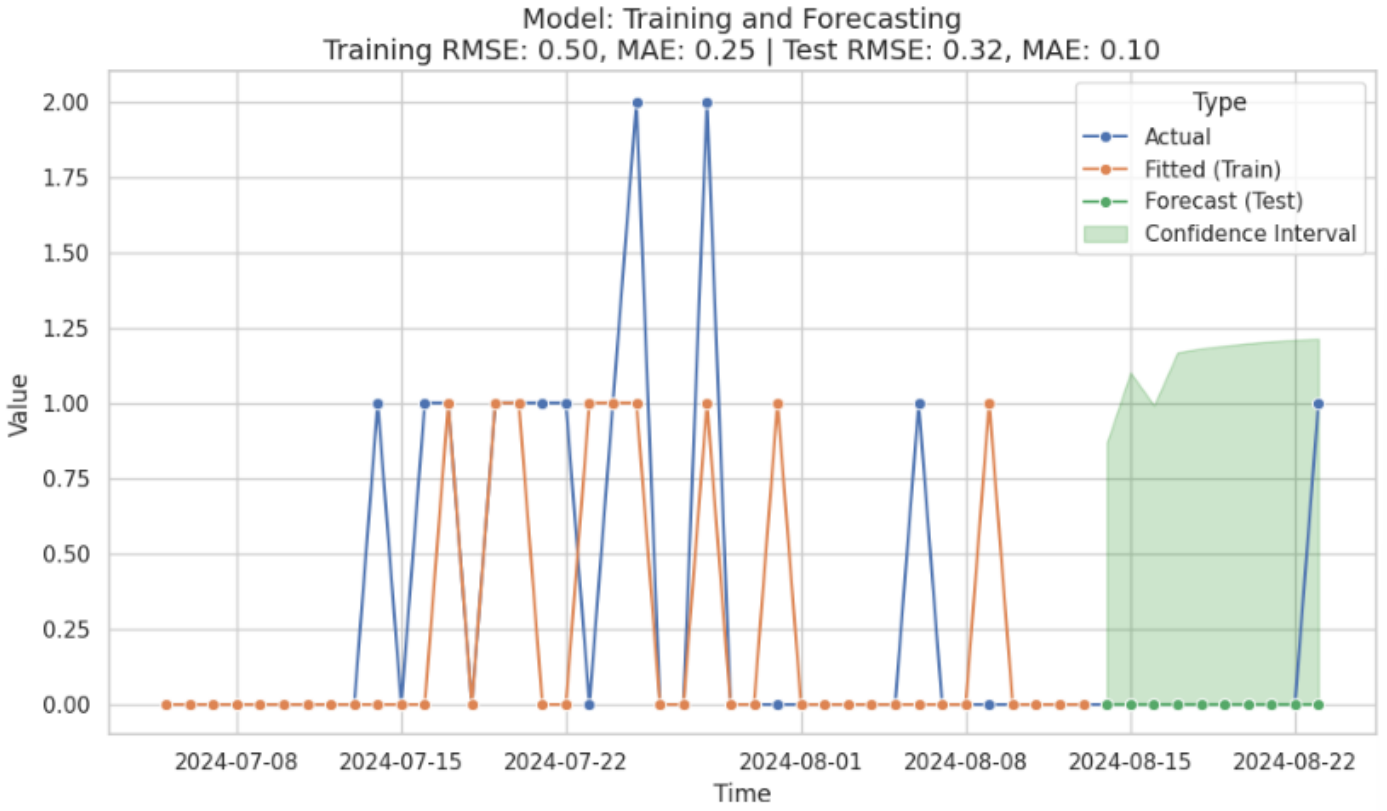
Ljung-Box (L1) (Q):	0.47	**Jarque-Bera (JB):**	17.37
Prob(Q):	0.49	**Prob(JB):**	0.00
Heteroskedasticity (H):	0.63	**Skew:**	-0.80
Prob(H) (two-sided):	0.41	**Kurtosis:**	5.81

Sigma2 indica la varianza dell'errore, cioè quanto i residui (le differenze tra i valori osservati e quelli previsti) sono distribuiti.

I test diagnostici, come quello sull'**eteroschedasticità**, verificano se la dispersione dei residui rimane costante nel tempo. Un valore di 0,63 suggerisce che i residui sono omoschedastici, ovvero la loro varabilità non varia nel tempo.

Inoltre, sono riportati anche i valori di **skew** (asimmetria) e **kurtosis** (curtosi), che descrivono la forma della distribuzione dei residui. Un valore di skew negativo (-0,80) indica che la distribuzione è leggermente inclinata verso sinistra, mentre un valore di kurtosis di 5,81 suggerisce che i residui seguono una distribuzione con code più spesse rispetto a una gaussiana.

Nuove catture



SARIMAX Results

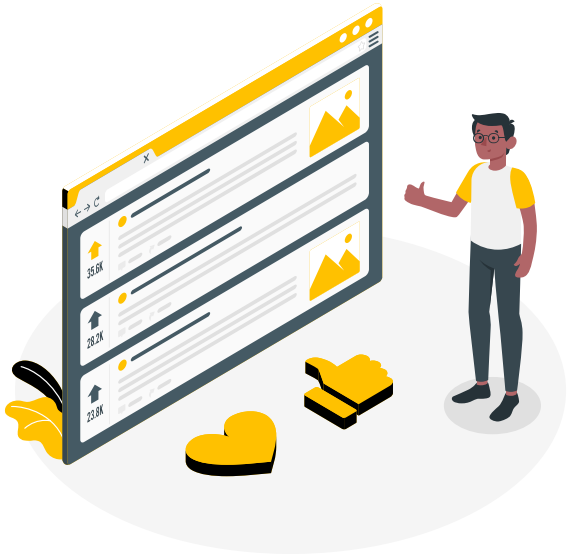
Dep. Variable:	y	No. Observations:	40
Model:	SARIMAX(1, 0, 4)	Log Likelihood	-26.009
Date:	Fri, 13 Dec 2024	AIC	64.018
Time:	12:04:39	BIC	74.151
Sample:	07-05-2024 - 08-13-2024	HQIC	67.682
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.9148	0.117	7.792	0.000	0.685	1.145
ma.L1	-0.8901	0.272	-3.277	0.001	-1.422	-0.358
ma.L2	0.0551	0.296	0.186	0.852	-0.525	0.635
ma.L3	0.6889	0.171	4.023	0.000	0.353	1.024
ma.L4	-0.4717	0.206	-2.295	0.022	-0.874	-0.069
sigma2	0.2027	0.049	4.176	0.000	0.108	0.298

Ljung-Box (L1) (Q):	0.01	Jarque-Bera (JB):	1.03
Prob(Q):	0.93	Prob(JB):	0.60
Heteroskedasticity (H):	0.63	Skew:	0.38
Prob(H) (two-sided):	0.42	Kurtosis:	3.23

VARIMAX

04



Vector Autoregressive Moving Average with Exogenous Variables

Questo approccio consente di modellare e prevedere variabili temporali, sfruttando sia le relazioni interne tra le variabili endogene, sia l'influenza dei fattori esterni rappresentati dalle variabili esogene.

Per Numero di insetti…

Variabili Endogene:

1. Numero di insetti (variabile target)
2. Temperatura
3. Umidità

Variabili Esogene:

1. Media_Mobile_4gg
2. PCA_Component
3. Nuove catture (per evento)

```
# Calcolo della media mobile a 4 giorni per 'Numero di insetti'
merged_df_copy['Media_Mobile_4gg'] = merged_df_copy['Numero di insetti'].rolling(window=4).sum()

# Riempimento dei valori NaN con 0
merged_df_copy = merged_df_copy.fillna(0)

# Selezione delle variabili per PCA
features_for_pca = merged_df_copy[['Media Temperatura', 'Media Umidità', 'Numero di insetti']]

# Standardizzazione delle variabili
scaler = StandardScaler()
scaled_features = scaler.fit_transform(features_for_pca)

# Riduzione dimensionale a una componente principale
pca = PCA(n_components=1)
pca_result = pca.fit_transform(scaled_features)
```

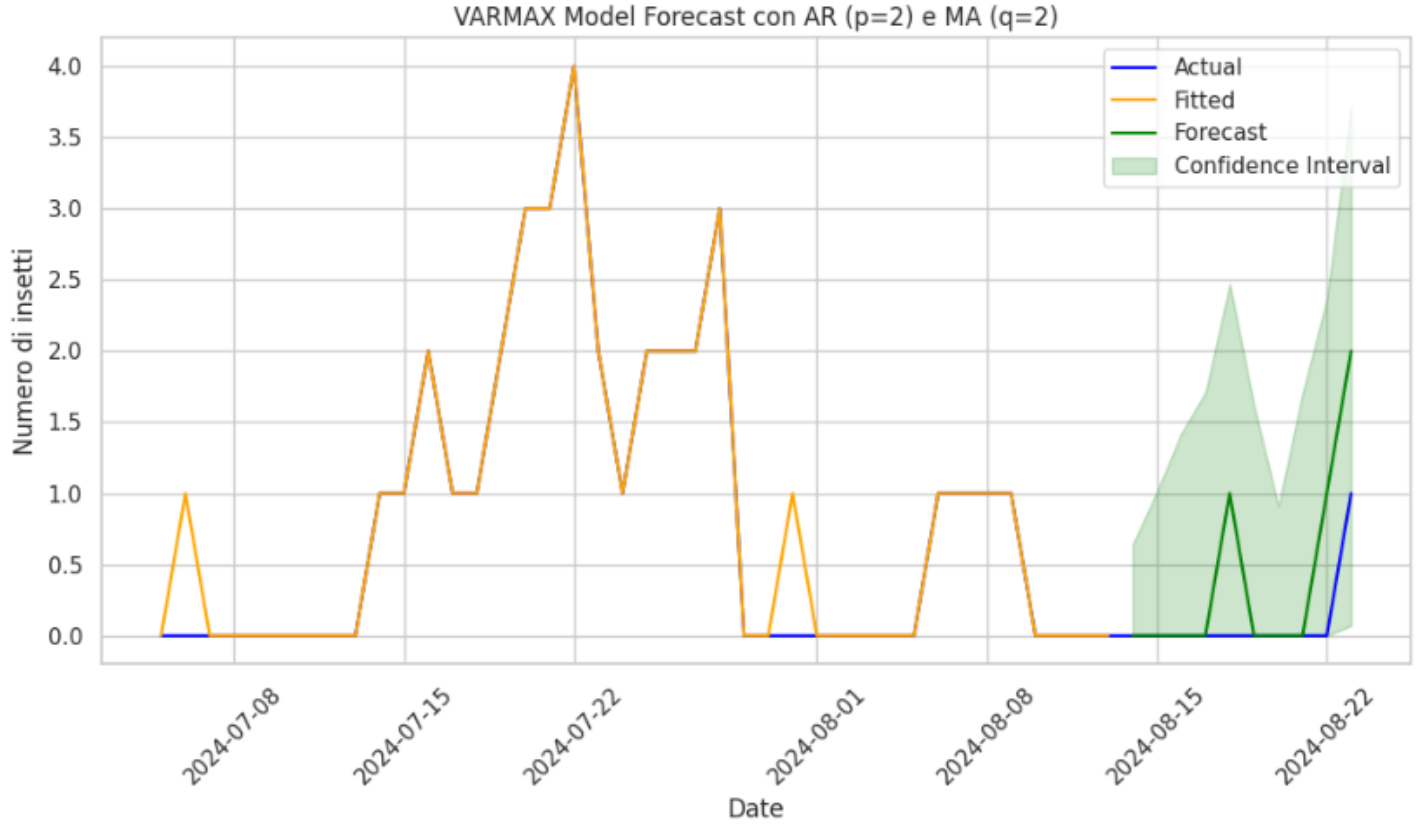
```
# Aggiunta della componente PCA al DataFrame
merged_df_copy['PCA_Component'] = pca_result
```

```
# Definizione delle variabili endogene ed esogene
endog = merged_df_copy[['Media Temperatura', 'Media Umidità', 'Numero di insetti']]  # Variabili endogene
exog = merged_df_copy[['Media_Mobile_4gg', 'PCA_Component', 'Nuove catture (per evento)']]  # Variabili esogene
```

DateTime	Media Temperatura	Media Umidità	Numero di insetti	Nuove catture (per evento)	Media_Mobile_4gg	PCA_Component
2024-07-05	25.777778	56.674444	0.0	0.0	0.0	-0.133474
2024-07-06	22.342500	72.247500	0.0	0.0	0.0	2.355880
2024-07-07	23.517917	76.728750	0.0	0.0	0.0	2.313096
2024-07-08	25.669167	69.143750	0.0	0.0	0.0	0.932927
2024-07-09	25.870000	53.647083	0.0	0.0	0.0	-0.415482

...

```
# Seleziona i parametri AR(p) e MA(q)
p = 2  # Ordine autoregressivo
q = 2  # Ordine media mobile

# Creazione e addestramento del modello VARMAX
varmax_model_1 = sm.tsa.VARMAX(endog_train, exog=exog_train, order=(p, q), trend='n').fit(disp=False)
```



Training RMSE: 0.22, MAE: 0.05
Test RMSE: 0.55, MAE: 0.30

Nuove catture

Variabili Endogene:

1. **Nuove catture (per evento)** (variabile target)
2. **Media Temperatura**
3. **Media Umidità**

Variabili Esogene:

1. **Media_Mobile_4gg**
2. **PCA_Component**
3. **Numero di insetti**

```
# Calcolo della media mobile a 4 giorni per 'Nuove catture (per evento)'
merged_df_copy['Media_Mobile_4gg'] = merged_df_copy['Nuove catture (per evento)'].rolling(window=4).sum()

# Impostazione della somma iniziale per i primi 3 valori
initial_mean = merged_df_copy['Nuove catture (per evento)'].iloc[:3].sum()
merged_df_copy['Media_Mobile_4gg'].iloc[:3] = initial_mean

# Selezione delle variabili per PCA
features_for_pca = merged_df_copy[['Media Temperatura', 'Media Umidità', 'Nuove catture (per evento)']]

# Standardizzazione delle variabili per PCA
scaler = StandardScaler()
scaled_features = scaler.fit_transform(features_for_pca)

# Riduzione dimensionale a una componente principale
pca = PCA(n_components=1)
pca_result = pca.fit_transform(scaled_features)

# Aggiunta della componente PCA al DataFrame
merged_df_copy['PCA_Component'] = pca_result
```

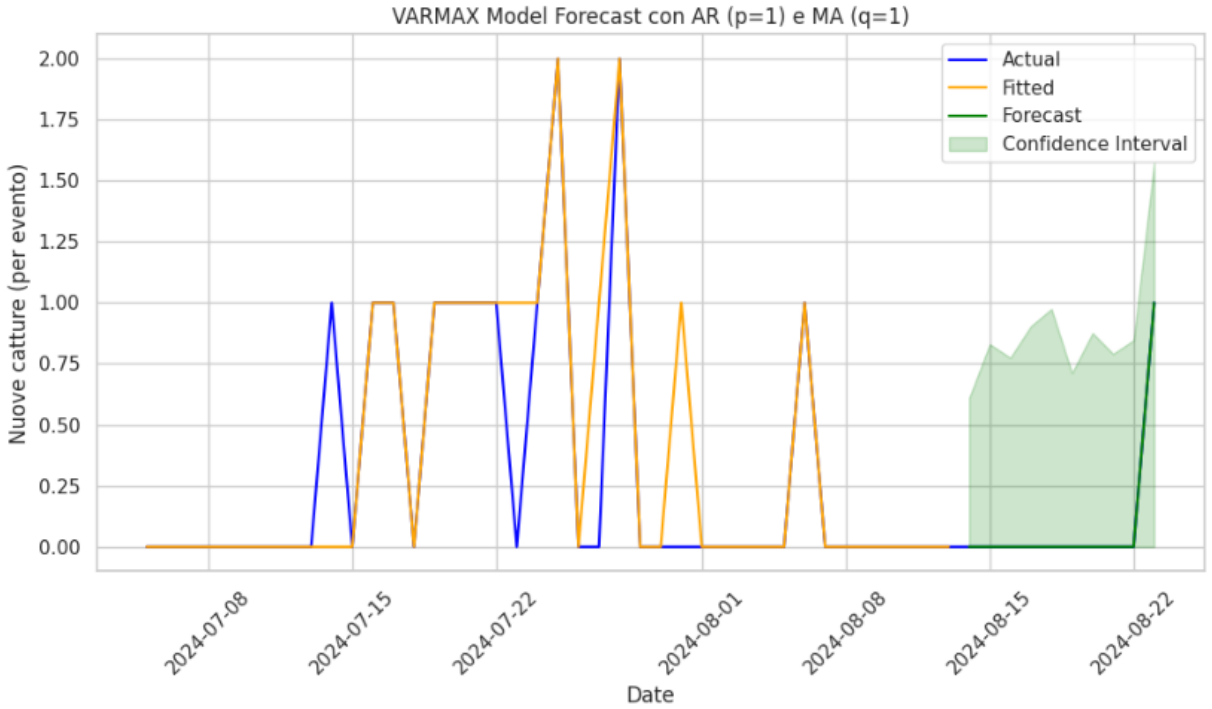
```
# Aggiunta della componente PCA al DataFrame
merged_df_copy['PCA_Component'] = pca_result
```

```
# Separazione tra variabili endogene ed esogene
endog = merged_df_copy[['Media Temperatura', 'Media Umidità', 'Nuove catture (per evento)']]
exog = merged_df_copy[['Media_Mobile_4gg', 'PCA_Component', 'Numero di insetti']]
```

DateTime	Media Temperatura	Media Umidità	Numero di insetti	Nuove catture (per evento)	Media_Mobile_4gg	PCA_Component
2024-07-05	25.777778	56.674444	0.0	0.0	0.0	0.120930
2024-07-06	22.342500	72.247500	0.0	0.0	0.0	-2.367527
2024-07-07	23.517917	76.728750	0.0	0.0	0.0	-2.323022
2024-07-08	25.669167	69.143750	0.0	0.0	0.0	-0.942967
2024-07-09	25.870000	53.647083	0.0	0.0	0.0	0.402374

...

```
# Parametri AR(p) e MA(q)
p2 = 1  # Parametro AR (modifica come necessario)
q2 = 1  # Parametro MA (modifica come necessario)

# Creazione e addestramento del modello VARMAX
varmax_model_2 = sm.tsa.VARMAX(endog_train, exog=exog_train, order=(p2, q2), trend='n').fit(disp=False)
```




```
Training RMSE: 0.32, MAE: 0.10
Test RMSE: 0.00, MAE: 0.00
```

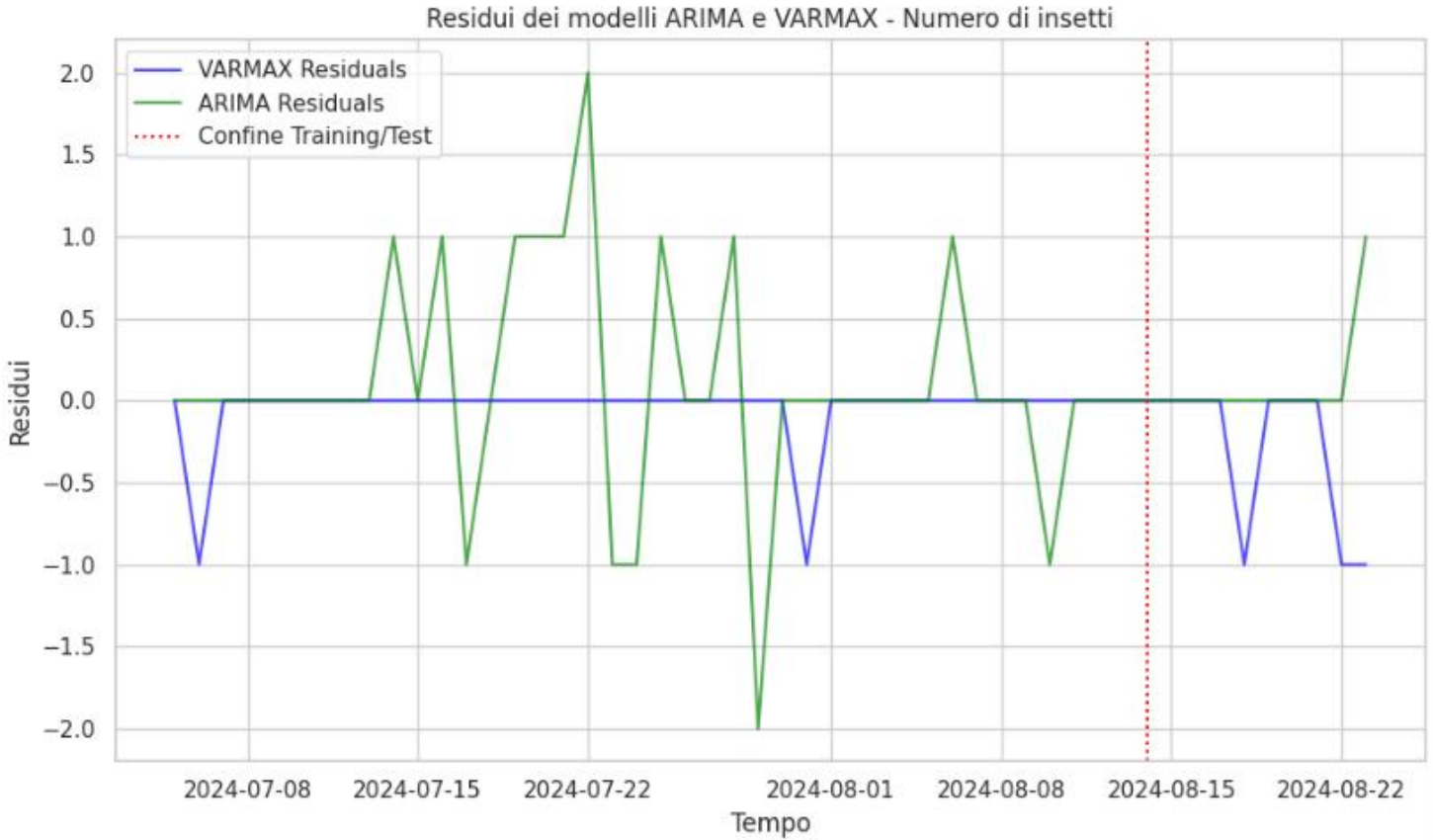
Confronto tra ARIMA e VARMAX

Per i set di training e testing sono stati calcolati:

- *Residui*: differenza tra i valori osservati e quelli previsti (errori di previsione).
- *RMSE* (*Root Mean Squared Error*): evidenzia gli errori più grandi calcolando la radice della media degli errori quadratici.
- *MAE* (*Mean Absolute Error*): media degli errori in valore assoluto, senza considerare il segno.

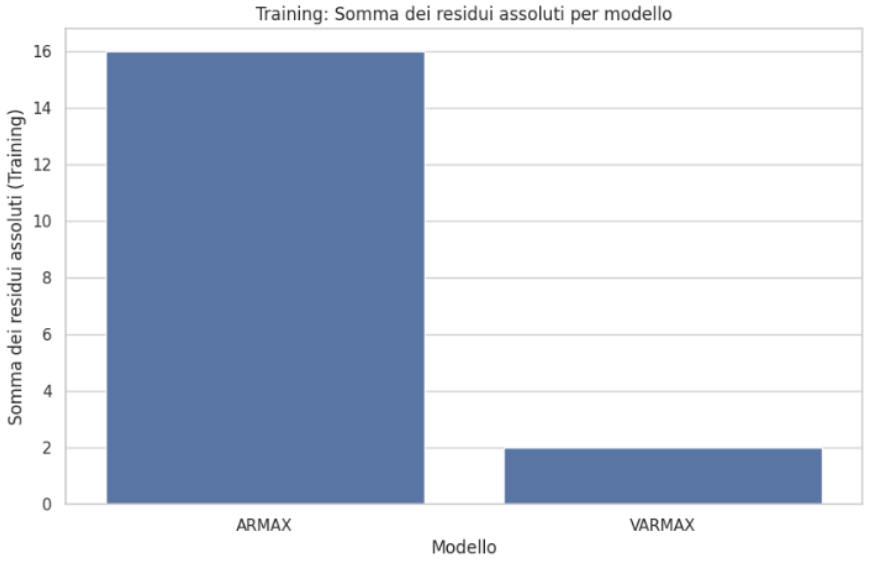
```
# Calcolo dei residui per il training set e il test set
residuals_train_arimax_1 = y_train - fitted_values_train  # Errori training
residuals_test_arimax_1 = y_test - forecasted_values_test  # Errori test

# Calcolo delle metriche di valutazione per il training set
rmse_train_arima_1 = np.sqrt(mean_squared_error(y_train, fitted_values_train))  # RMSE training
mae_train_arima_1 = mean_absolute_error(y_train, fitted_values_train)  # MAE training

# Calcolo delle metriche di valutazione per il test set
rmse_test_arima_1 = np.sqrt(mean_squared_error(y_test, forecasted_values_test))  # RMSE test
mae_test_arima_1 = mean_absolute_error(y_test, forecasted_values_test)  # MAE test
```

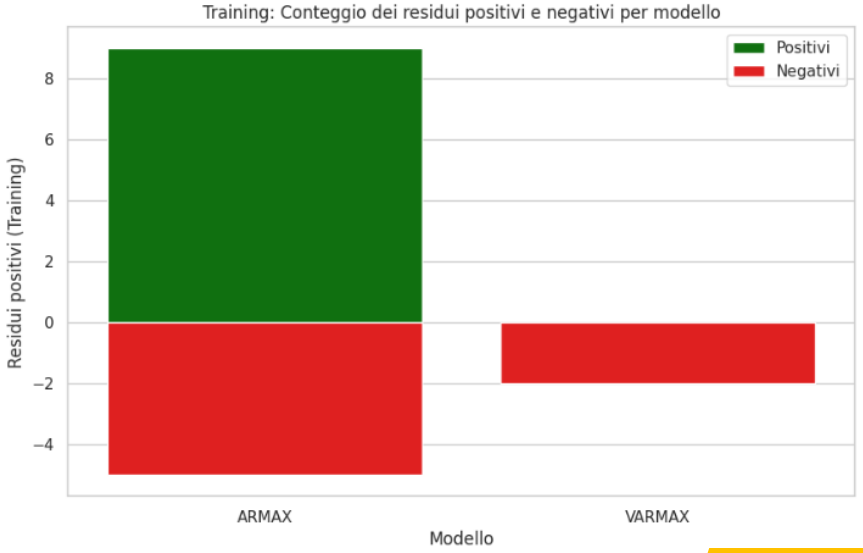
```
# Calcolo degli errori (RMSE e MAE) per training e test set
rmse_train_varima_1 = np.sqrt(mean_squared_error(endog_train['Numero di insetti'], fitted_values_train['Numero di insetti']))
mae_train_varima_1 = mean_absolute_error(endog_train['Numero di insetti'], fitted_values_train['Numero di insetti'])
rmse_test_varima_1 = np.sqrt(mean_squared_error(endog_test['Numero di insetti'], forecasted_values_test['Numero di insetti']))
mae_test_varima_1 = mean_absolute_error(endog_test['Numero di insetti'], forecasted_values_test['Numero di insetti'])

# Residui (errore tra valori reali e previsti)
residuals_train_varmax_1 = endog_train['Numero di insetti'] - fitted_values_train['Numero di insetti']
residuals_test_varmax_1 = endog_test['Numero di insetti'] - forecasted_values_test['Numero di insetti']
```

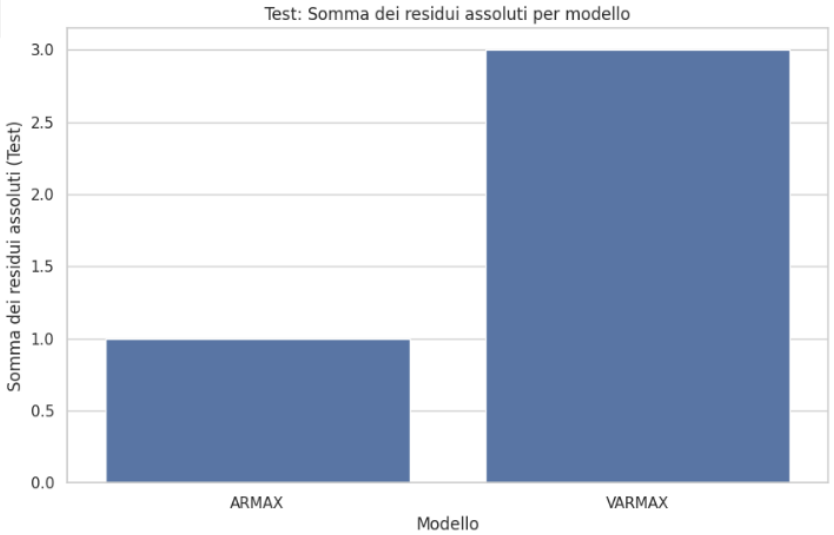

Residui dei modelli ARIMA e VARMAX - Numero di insetti
VARMAX Residuals
ARIMA Residuals
Confine Training/Test
Residui
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
2024-07-08
2024-07-15
2024-07-22
2024-08-01
2024-08-08
2024-08-15
2024-08-22
Tempo

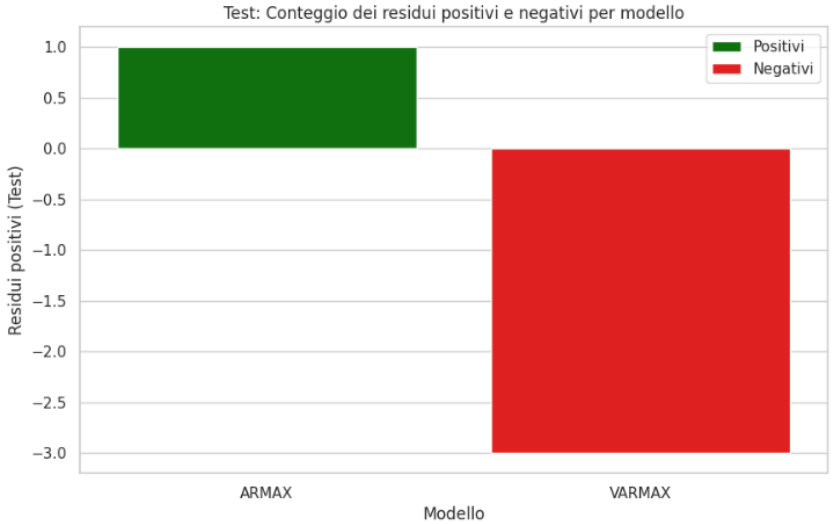
La somma dei residui assoluti aiuta a capire quanto bene il modello si adatta ai dati: minore è la somma, migliore è il modello.

Il *conteggio dei residui positivi e negativi*, invece, mostra se c'è un equilibrio tra i due: un buon modello dovrebbe avere un numero simile di residui positivi e negativi. Se uno dei due è molto più numeroso, potrebbe indicare che il modello ha un bias o non si adatta perfettamente ai dati.

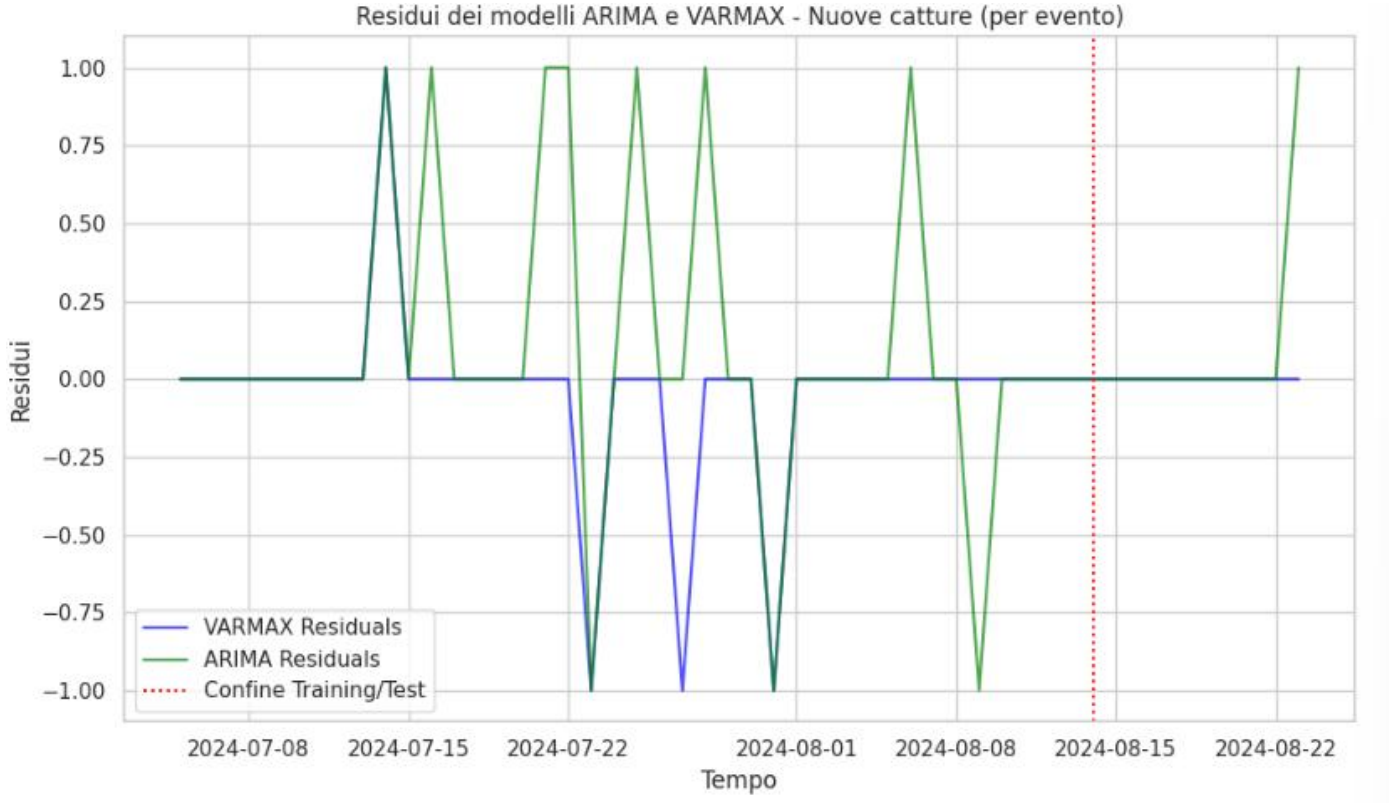


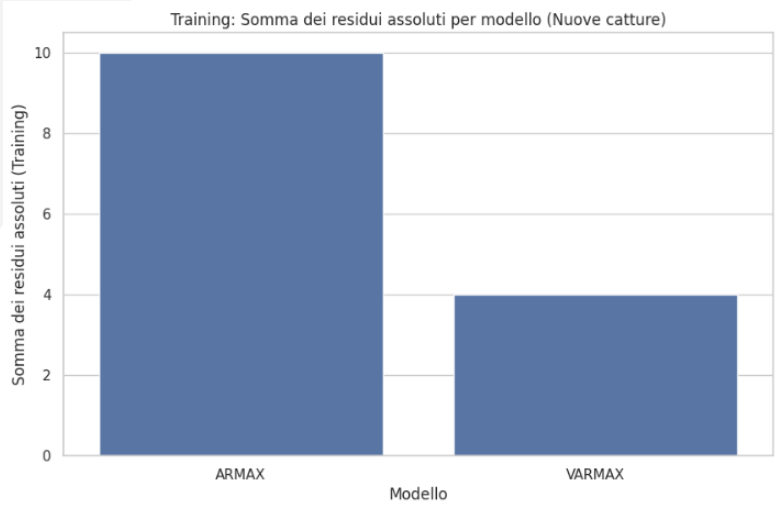


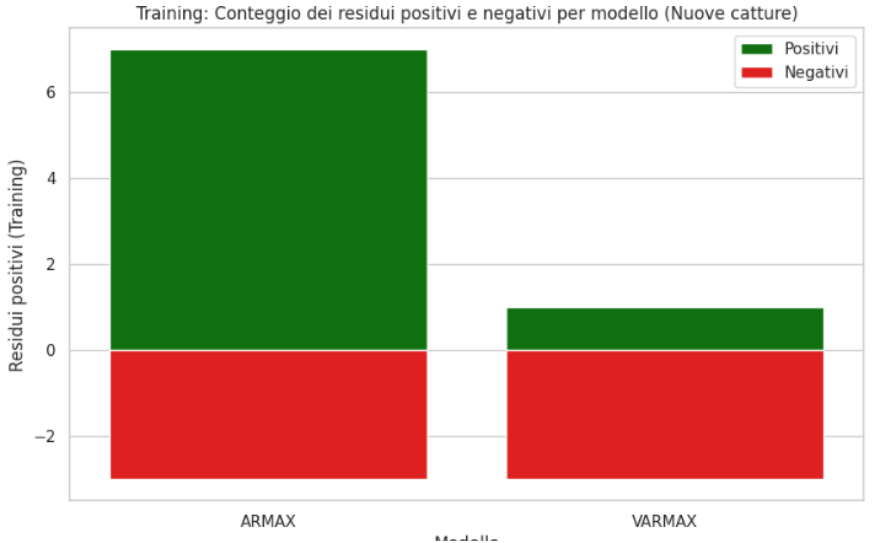

Test: Somma dei residui assoluti per modello
Somma dei residui assoluti (Test)
3.0
2.5
2.0
1.5
1.0
0.5
0.0
ARMAX
VARMAX
Modello

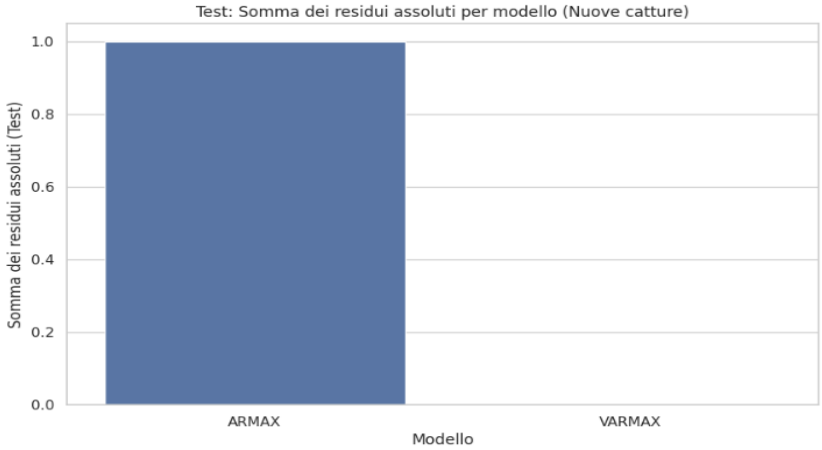

Test: Conteggio dei residui positivi e negativi per modello
Positivi
Negativi
Residui positivi (Test)
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
−3.0
ARMAX
VARMAX
Modello

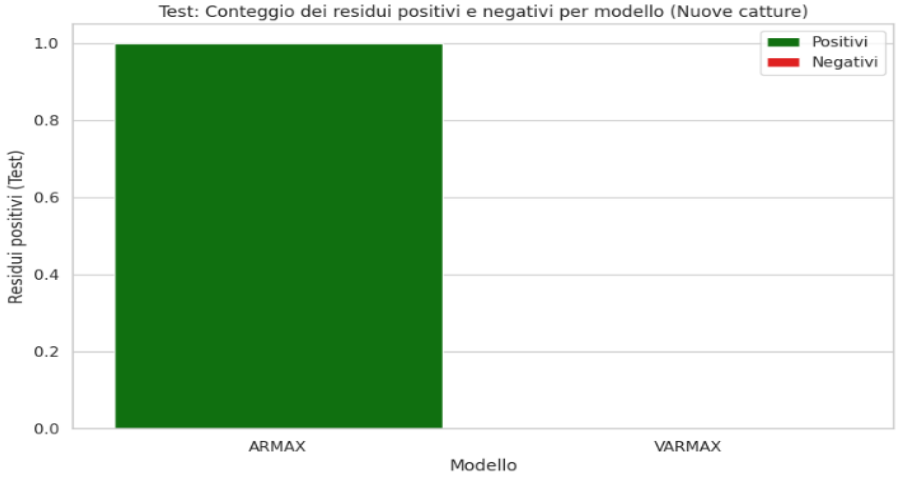
Confronto tra il modello ARIMA e VARMAX per Nuove catture




Training: Somma dei residui assoluti per modello (Nuove catture)
Somma dei residui assoluti (Training)
0
2
4
6
8
10
ARMAX
VARMAX
Modello


Training: Conteggio dei residui positivi e negativi per modello (Nuove catture)
Positivi
Negativi
Residui positivi (Training)
−2
0
2
4
6
ARMAX
VARMAX


Test: Somma dei residui assoluti per modello (Nuove catture)
Somma dei residui assoluti (Test)
0.0
0.2
0.4
0.6
0.8
1.0
ARMAX
VARMAX
Modello


Test: Conteggio dei residui positivi e negativi per modello (Nuove catture)
Positivi
Negativi
Residui positivi (Test)
0.0
0.2
0.4
0.6
0.8
1.0
ARMAX
VARMAX
Modello

Ensemble learning

05

L'ensemble combina più modelli di previsione per migliorare l'accuratezza e la robustezza, sfruttando i punti di forza di ciascun modello. In particolare, riduce gli errori individuali e contrastando fenomeni di overfitting o underfitting.

Modelli utilizzati:

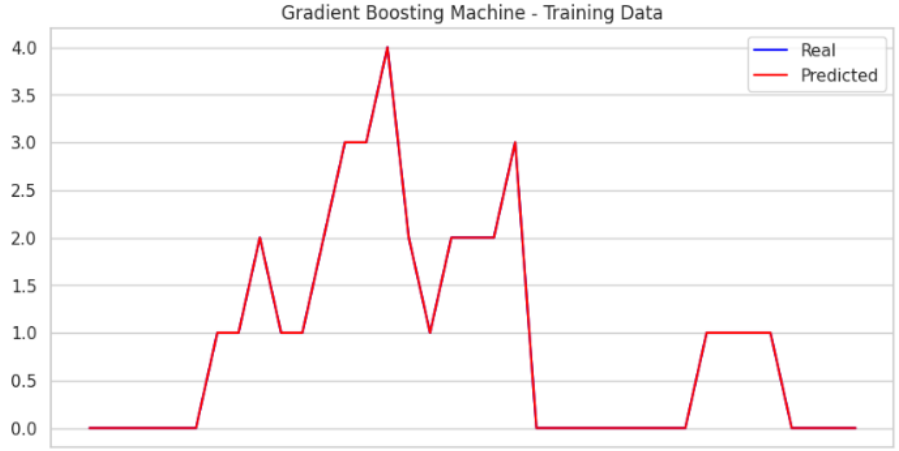
- *Gradient Boosting Regression* (*GBM*): costruisce iterativamente alberi decisionali, correggendo gli errori residui. Migliora progressivamente le previsioni concentrandosi sugli errori più difficili.
- *Random Forest Regressor*: aggrega le previsioni di una foresta di alberi decisionali. Riduce la varianza combinando le medie, garantendo stabilità e robustezza.

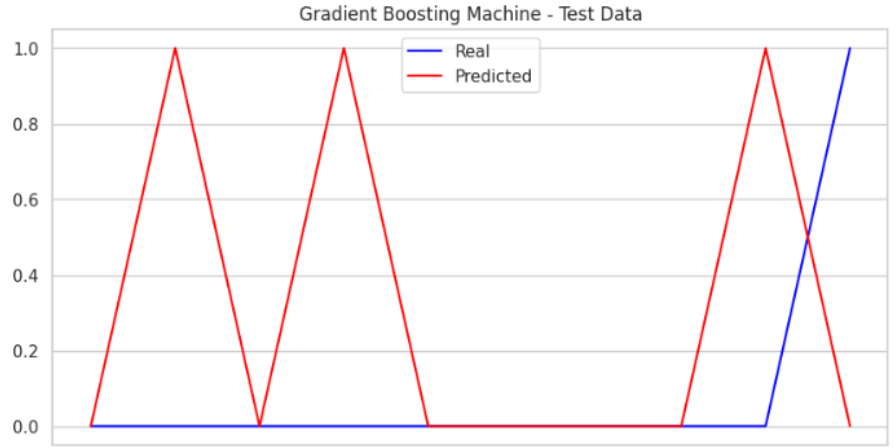
```python
def create_lagged_features(df, target_col, n_lags=5, exog_cols=[]):
    """Crea caratteristiche ritardate per la colonna target e lascia invariati i dati esogeni."""
    # Creazione dei lag per la colonna target
    for i in range(1, n_lags + 1):
        df[f'lag_{i}'] = df[target_col].shift(i)

    # Rimozione delle righe con valori nulli
    df.dropna(inplace=True)
    return df
```

```python
# Definizione dei modelli da allenare (Gradient Boosting e Random Forest)
models = {
    "Gradient Boosting Machine": GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, max_depth=3),
    "Random Forest": RandomForestRegressor(n_estimators=100, max_depth=3)
}
```

Numero di insetti - Gradient Boosting Machine

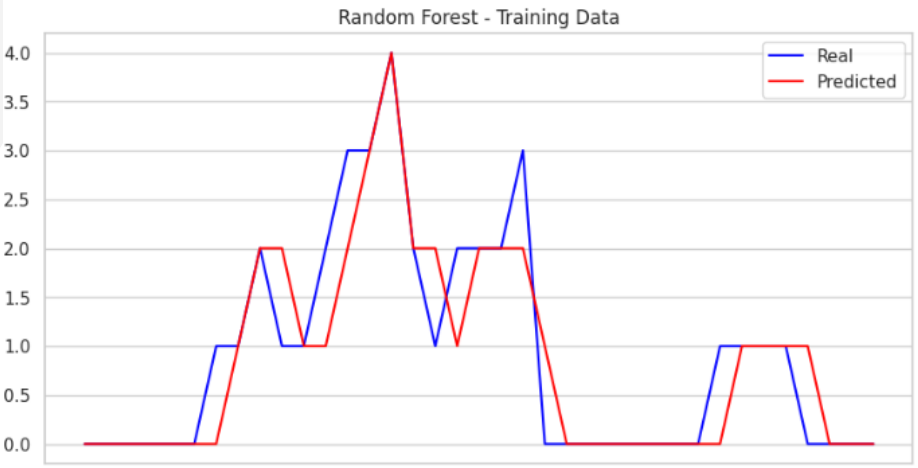
```
# Numero di lag da usare per il modello
n_lags = 3  # Modifica il valore se necessario

# Chiamata alla funzione per allenare i modelli e visualizzare i risultati per "Numero di insetti"
models, rmse_train_ensamble_1, mae_train_ensamble_1, rmse_test_ensamble_1, mae_test_ensamble_1, real_data = train_and_visualize(
    df_merged_copy, n_lags, exog_cols, target_col='Numero di insetti')
```

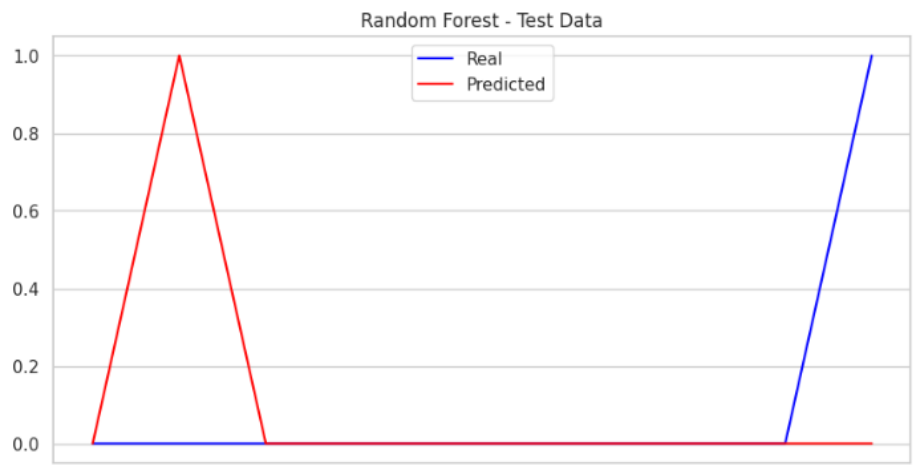




Training RMSE: 0.00, MAE: 0.00
Test RMSE: 0.63, MAE: 0.40

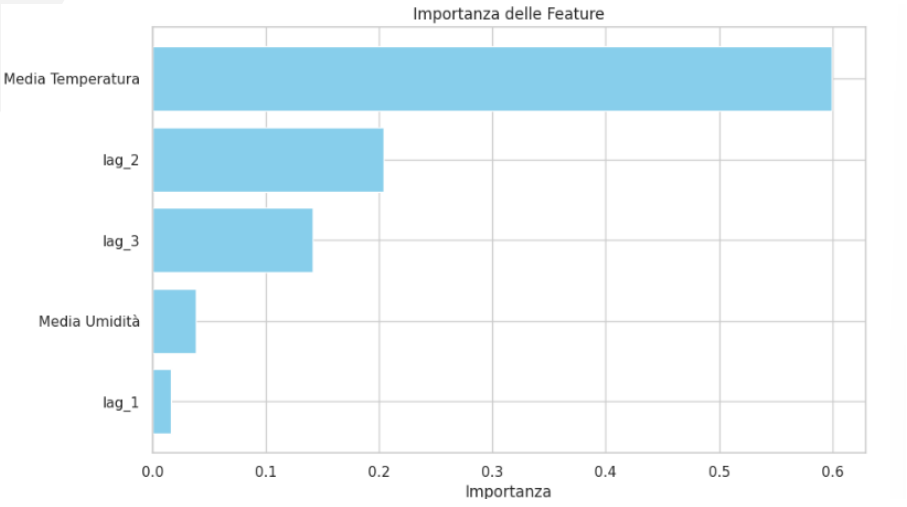
Numero di insetti - Random Forest



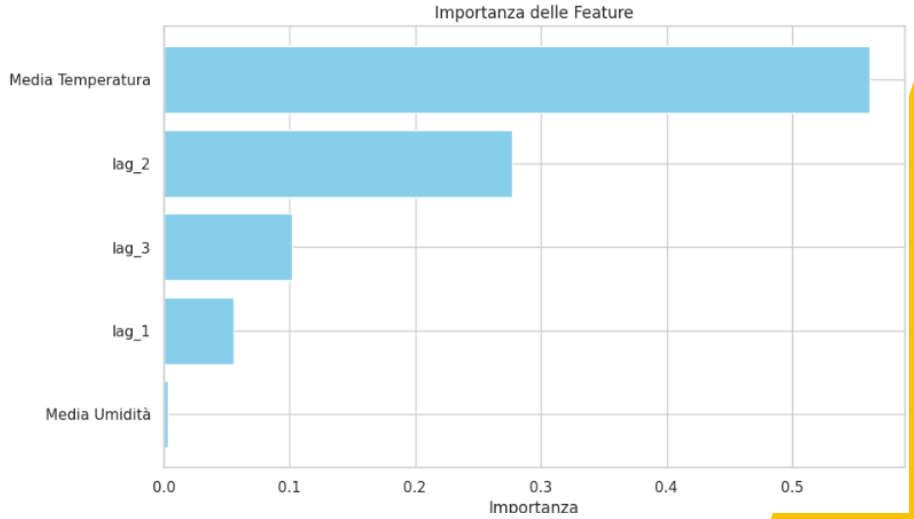



```
Training RMSE: 0.52, MAE: 0.27
Test RMSE: 0.45, MAE: 0.20
```

Numero di insetti - Importanza delle Feature

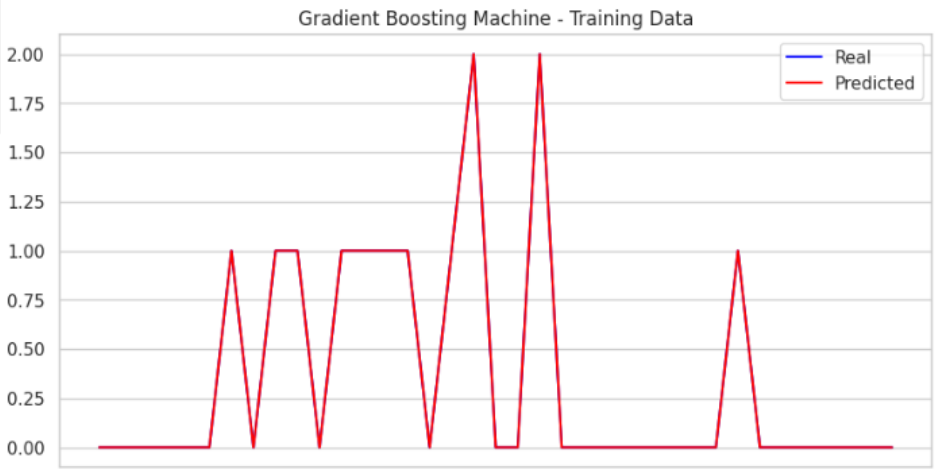


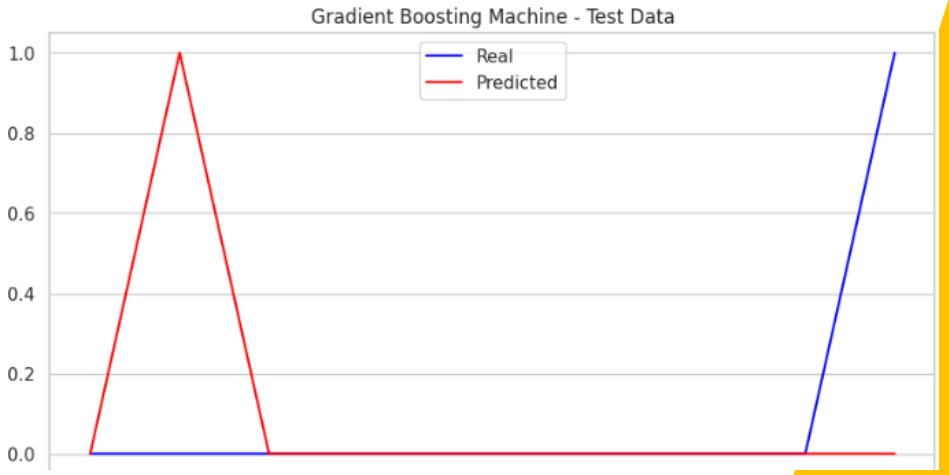
Random Forest



Gradient Boosting

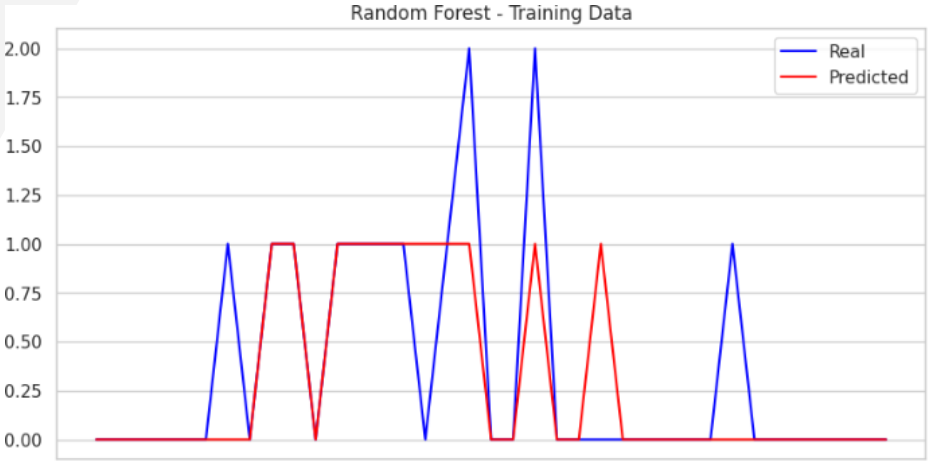
Nuove catture - Gradient Boosting





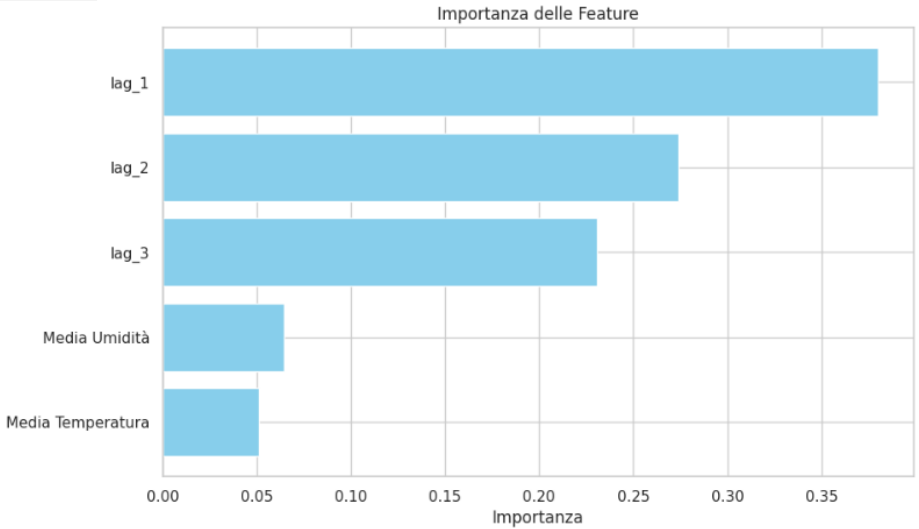

```
Training RMSE: 0.00, MAE: 0.00
Test RMSE: 0.45, MAE: 0.20
```

Nuove catture - Random Forest

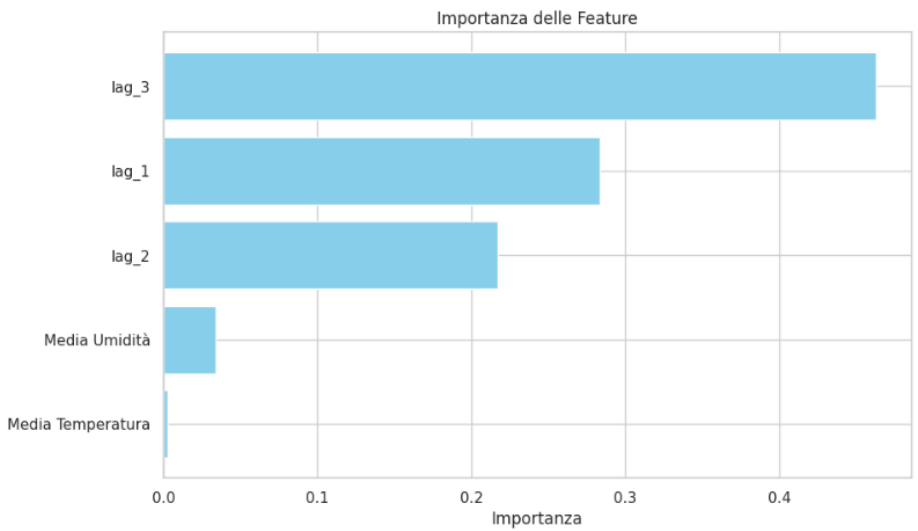



```
Training RMSE: 0.40, MAE: 0.16
Test RMSE: 0.32, MAE: 0.10
```

Nuove catture - Importanza delle Feature



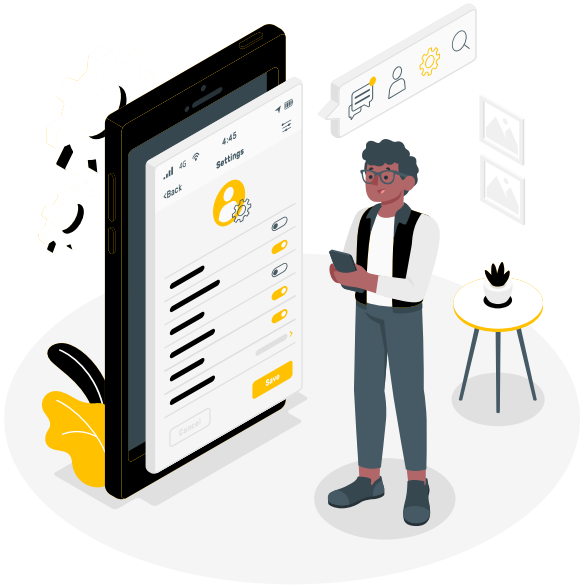
Random Forest



Gradient Boosting

Multi-Layer Perceptron

06

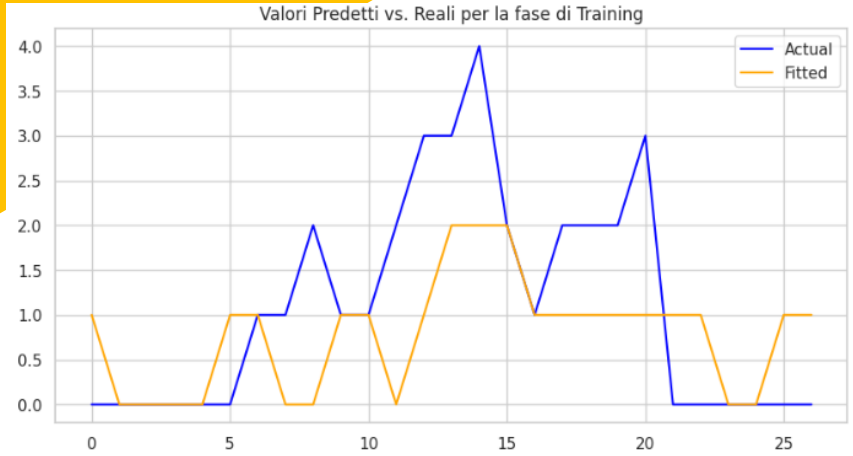


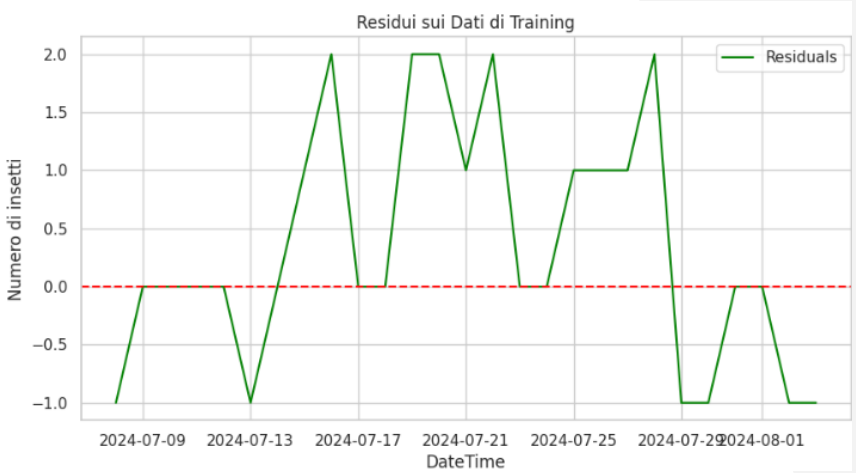
Il modello MLP utilizza come input una combinazione di *variabili laggate ed esogene*. Le variabili lag rappresentano i valori passati della variabile target, mentre le variabili esogene includono informazioni aggiuntive, come la media della temperatura e la media dell'umidità. In questo caso, sono stati creati tre lag della variabile target per catturare l'andamento temporale.

La rete neurale è composta da due strati completamente connessi (*dense layers*):

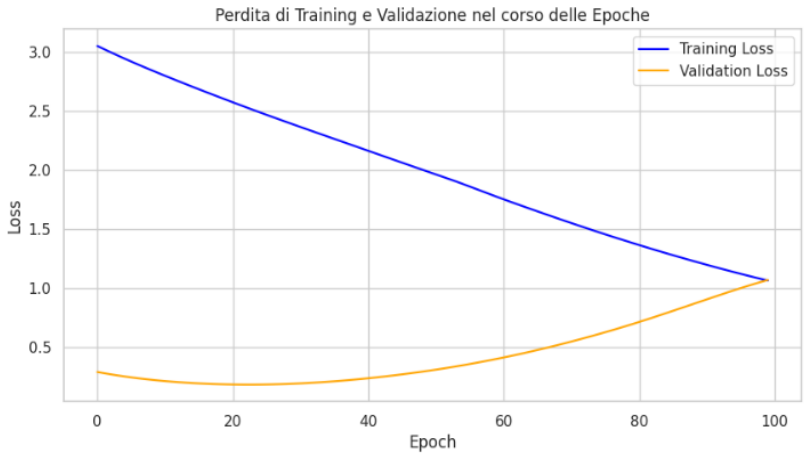
- Il primo strato ha 5 neuroni, progettato per elaborare una prima trasformazione dei dati in ingresso.
- Il secondo strato ha 30 neuroni, per catturare relazioni più complesse tra le variabili.

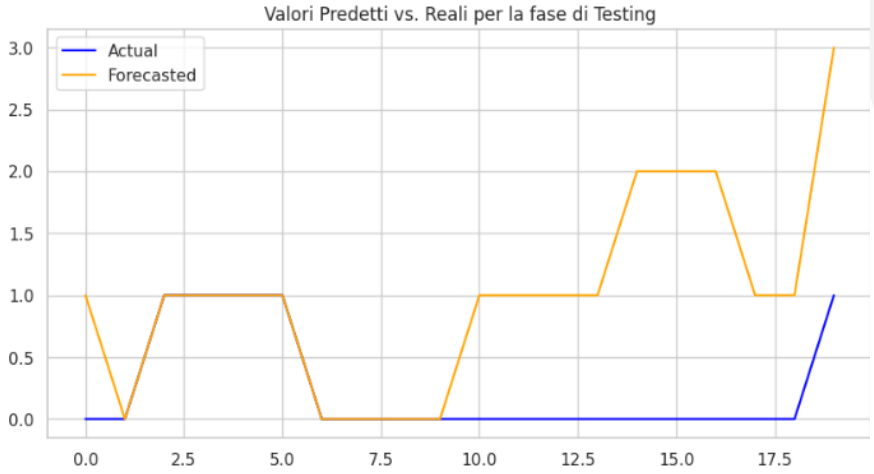
```
# Compila e addestra il modello
model_mlp.compile(optimizer='adam', loss='mean_squared_error')
history = model_mlp.fit(
    X_train_scaled, y_train,
    epochs=100, batch_size=32,
    validation_data=(X_test_scaled, y_test),
    verbose=0
)
```

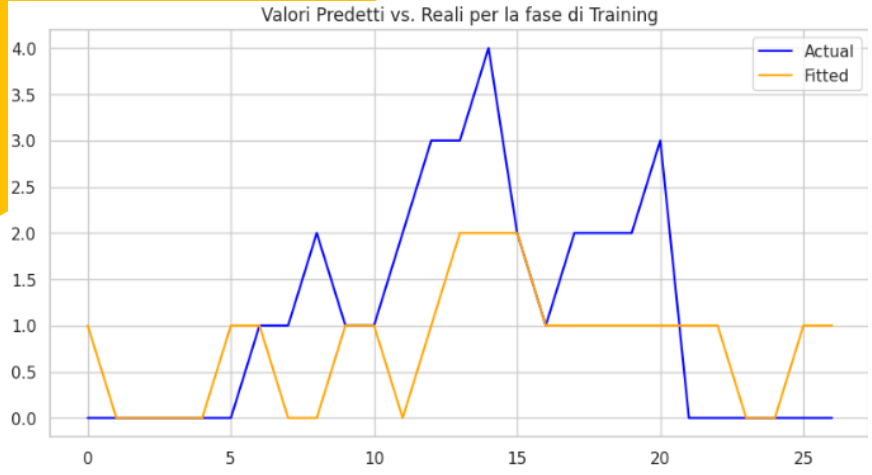


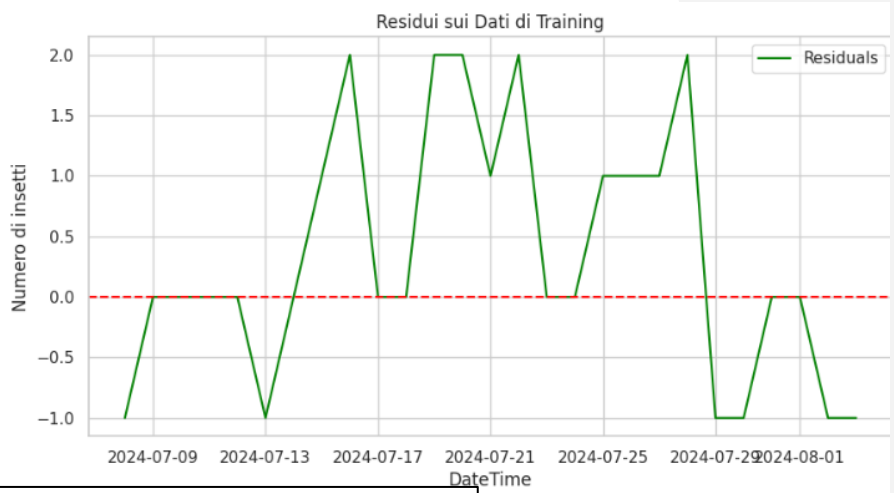


Numero di insetti → RMSE Training: 1.07, MAE Training: 0.78, RMSE Test: 1.07, MAE Test: 0.75

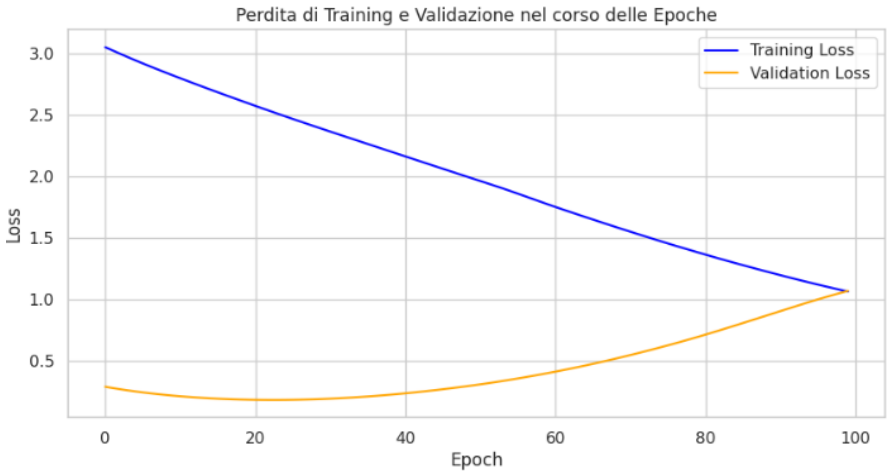


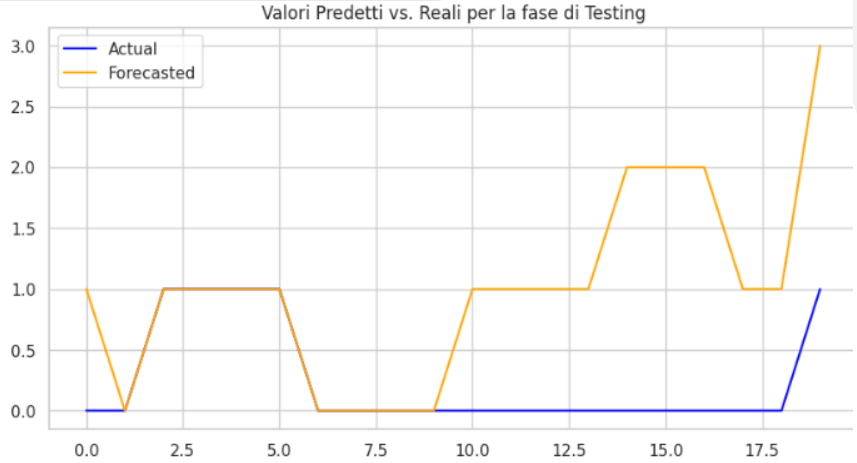






Nuove catture → RMSE Training: 1.07, MAE Training: 0.78, RMSE Test: 1.07, MAE Test: 0.75





Long Short-Term Memory

07

I dati sono stati preparati creando lag fino a 5 periodi per le variabili target (*Numero di insetti* e *Nuove catture*) e le variabili esogene (*Temperatura* e *Umidità*), al fine di catturare le dipendenze temporali includendo informazioni passate.

I dati sono stati organizzati in *X_list*, una lista di array per i ritardi temporali, e combinati in una struttura tridimensionale tramite *np.stack*:

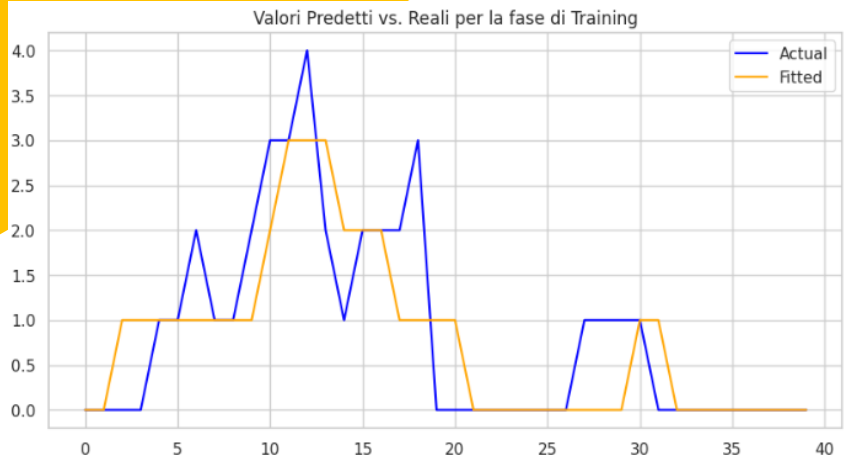
(n. osservazioni, n. lag, n. feature)

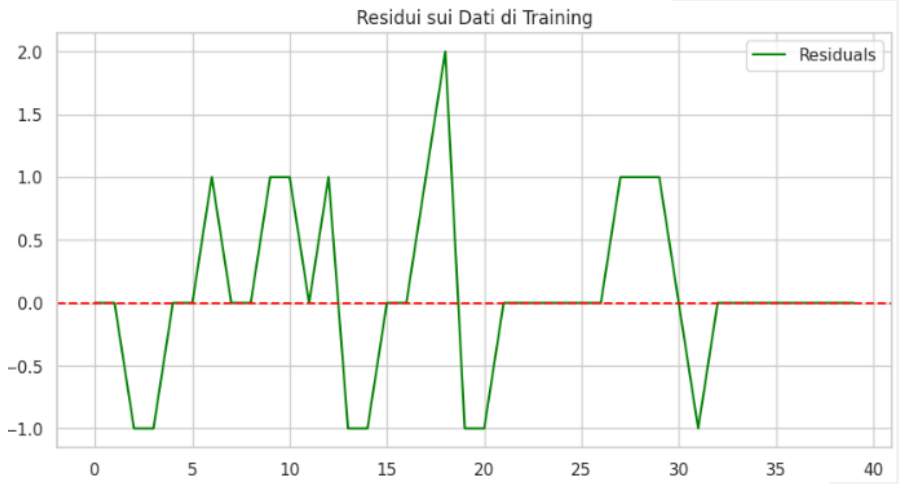
compatibile con il modello LSTM.

Divisi in set di training e test, il modello LSTM include:

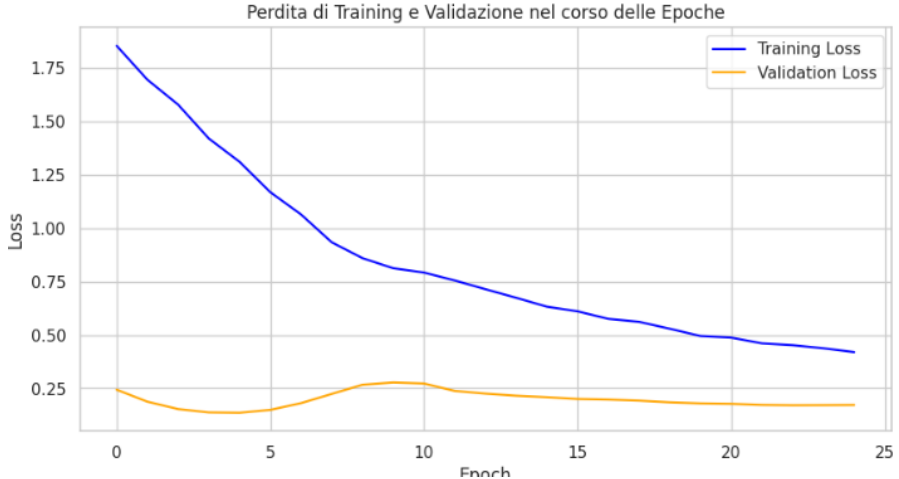
- Un layer di 50 unità con funzione di attivazione *ReLU* per non linearità.
- Un layer denso per la previsione della specifica variabile target.

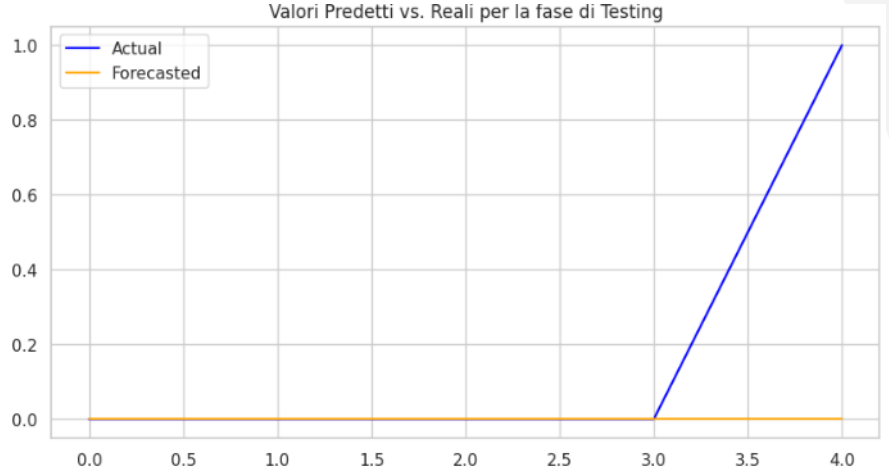
```
# Definizione del modello LSTM
inputs = tf.keras.Input(shape=(X_train_scaled.shape[1], X_train_scaled.shape[2]))
x = tf.keras.layers.LSTM(50, activation='relu')(inputs)
outputs = tf.keras.layers.Dense(1)(x)
model_lstm = tf.keras.Model(inputs=inputs, outputs=outputs)
model_lstm.compile(optimizer='adam', loss='mean_squared_error')

# Addestramento del modello
history = model_lstm.fit(
    X_train_scaled, y_train,
    epochs=25,
    batch_size=8,
    validation_data=(X_test_scaled, y_test),
    verbose=0
)
```

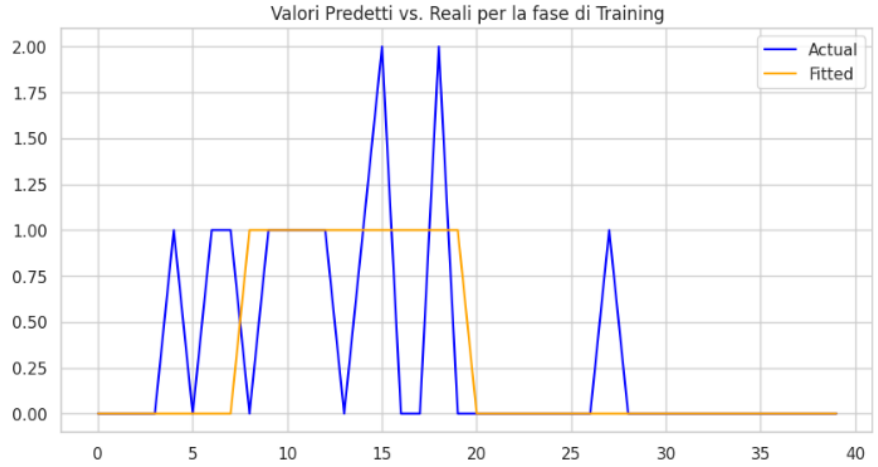




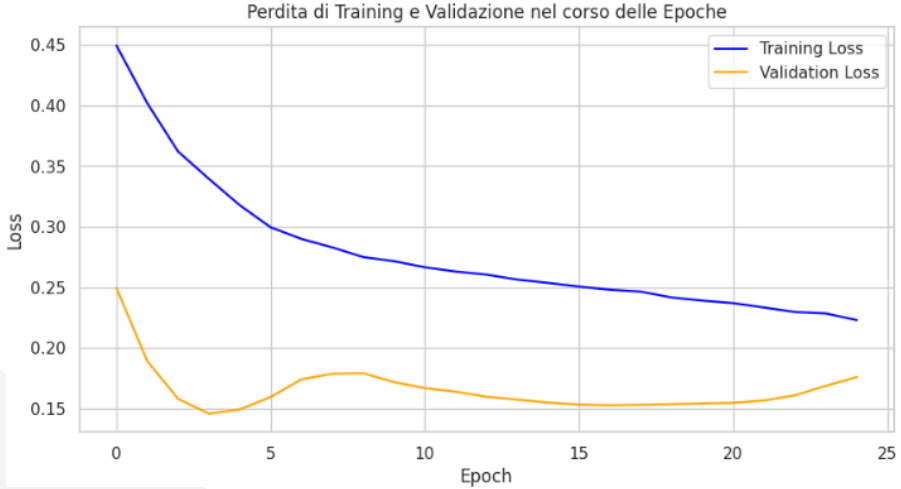
Numero di insetti → RMSE Training: 0.69, MAE Training: 0.42, RMSE Test: 0.45, MAE Test: 0.20

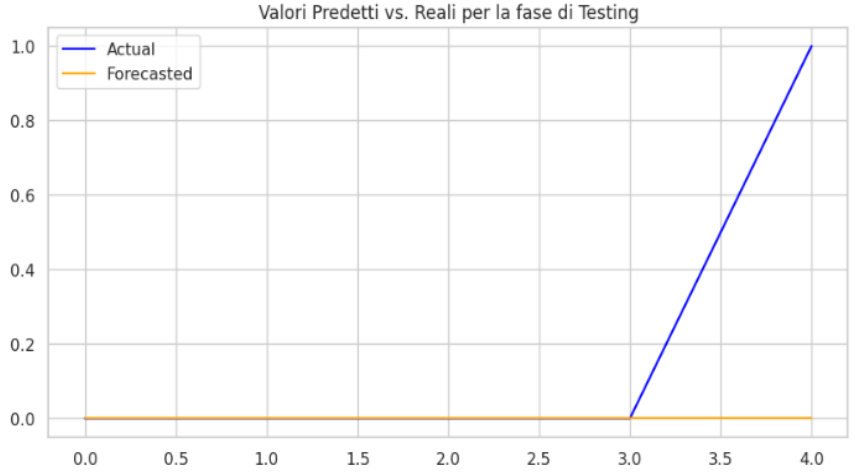






Nuove catture → RMSE Training: 0.52, MAE Training: 0.28, RMSE Test: 0.45, MAE Test: 0.20

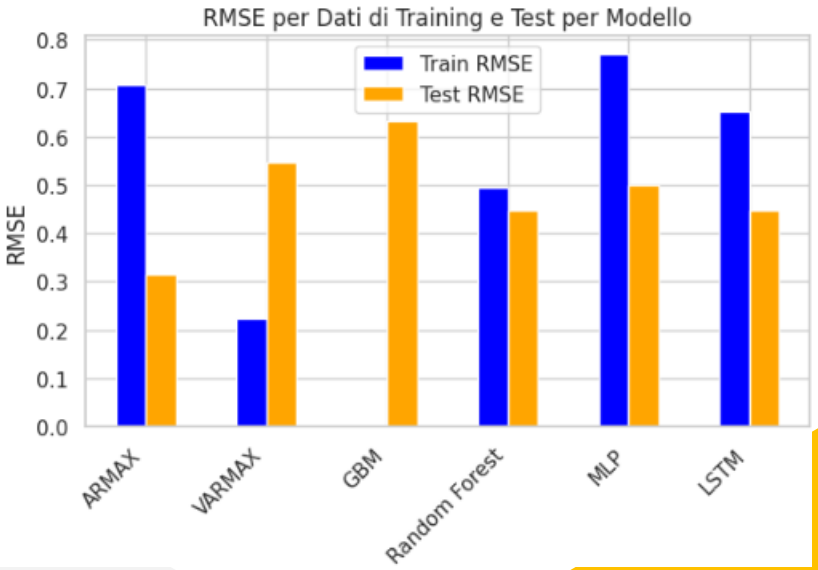


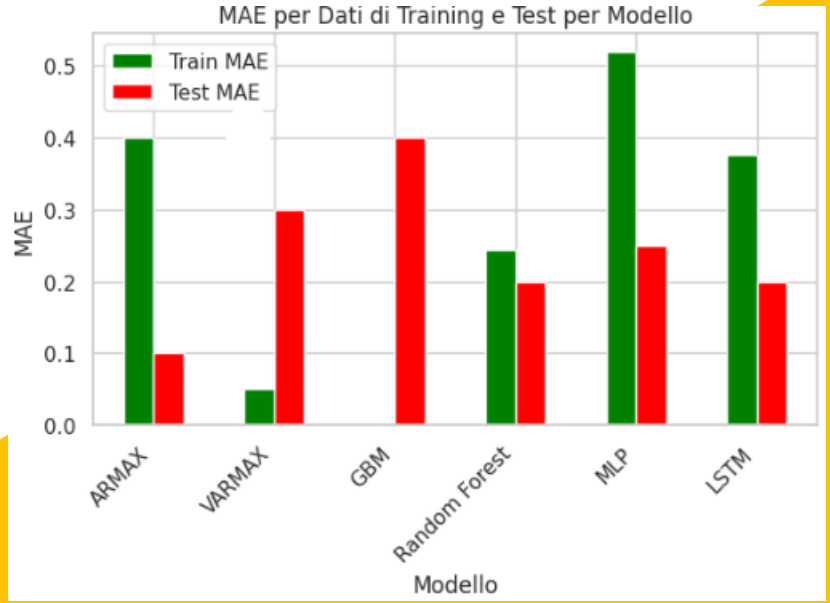


Confronto tra tutti i modelli

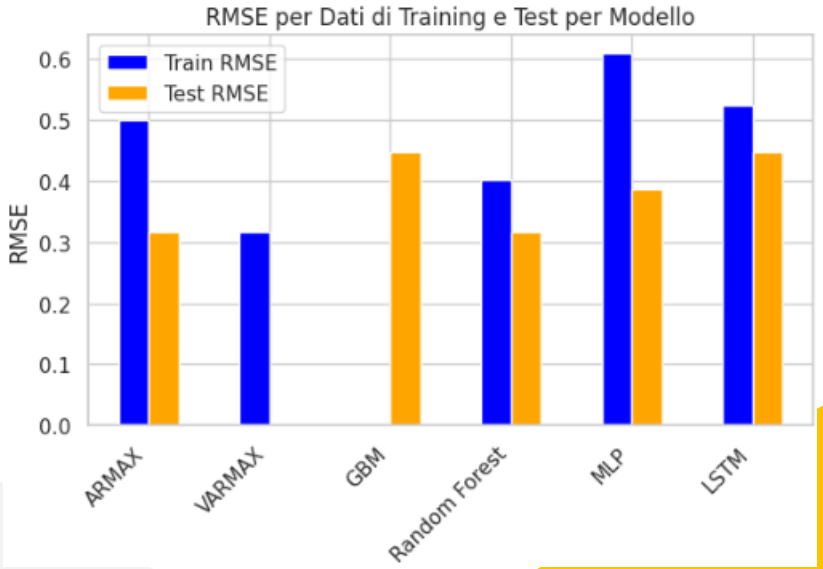
08

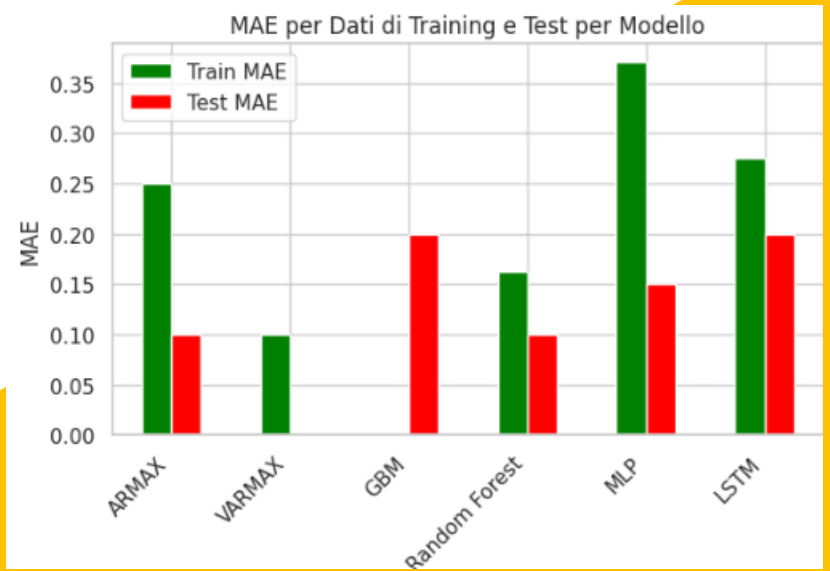
Numero di insetti



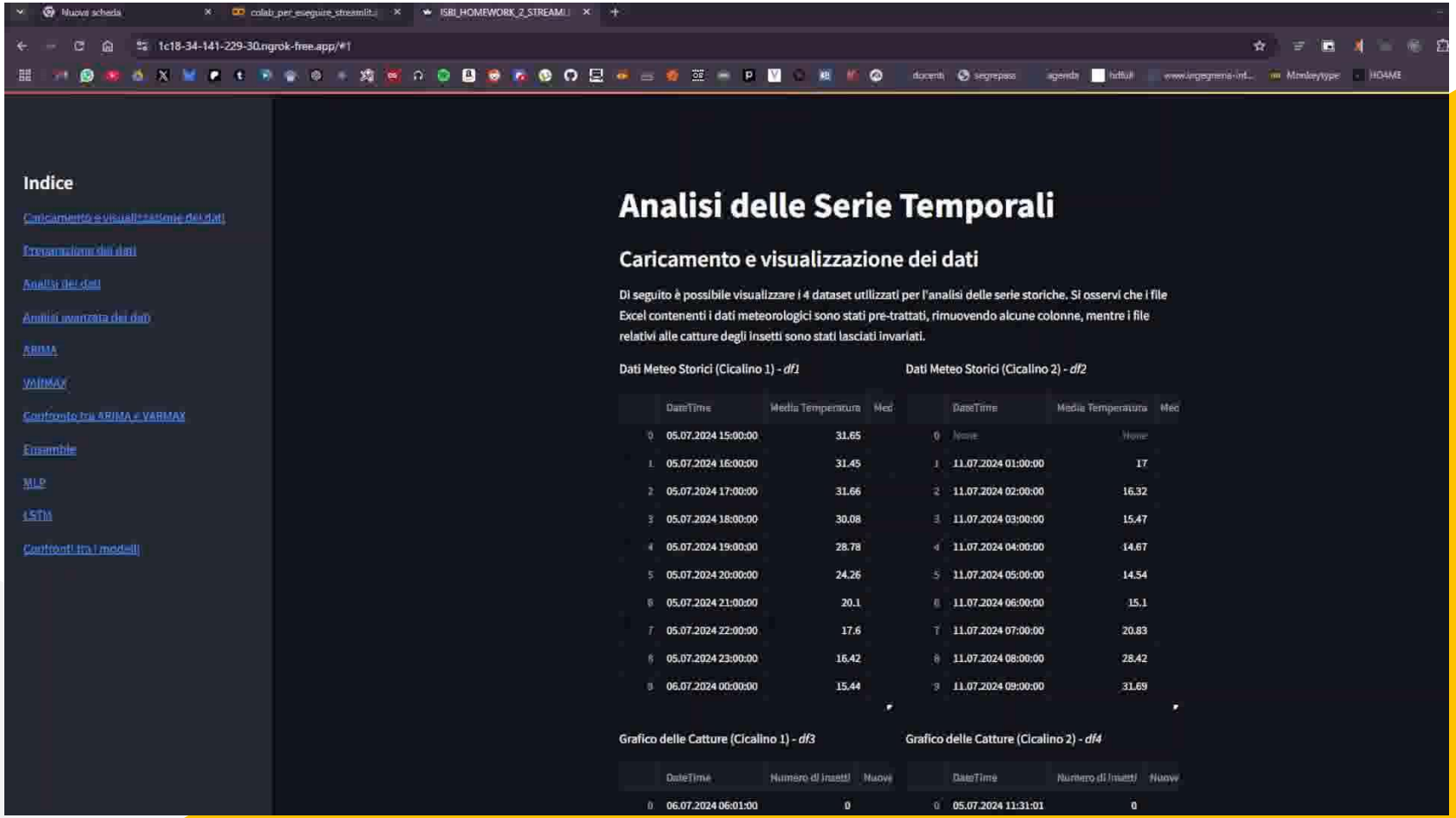


Nuove catture





Pagina web realizzata con Streamlit



Grazie mille per l’attenzione!