



Scuola Politecnica e delle Scienze di Base  
Corso di Laurea Magistrale in Ingegneria Informatica

*Sviluppo di un notebook in Google Colab per il trattamento dei dati -  
"daily climate change in Dehli,  
India"*

Anno Accademico 2023/2024

Professore  
**Flora Amato**

Studenti  
**Gaetano Saviano matr. M63001502**  
**Francesco Della Valle matr. M63001500**

# Contents

<b>1</b>	<b>Notebook Google Colab</b>	<b>1</b>
1.1	Traccia . . . . .	1
1.2	Google Colab . . . . .	2
<b>2</b>	<b>Dataset</b>	<b>4</b>
2.1	Descrizione del Dataset . . . . .	4
2.2	Descrizione delle features . . . . .	4
2.3	Obiettivo . . . . .	5
<b>3</b>	<b>Analisi e visualizzazione dei dati</b>	<b>6</b>
3.1	Fase Preliminare . . . . .	6
3.1.1	Montaggio di Google Drive . . . . .	6
3.1.2	Importazione Librerie . . . . .	7
3.1.3	Lettura dei Dati e Statistiche Descrittive . . .	8
3.2	Visualizzazioni dei dati . . . . .	9
3.2.1	Visualizzazione dei dati in sfaccettature . . . .	9
3.2.2	Una visione più profonda . . . . .	11
3.2.3	Missing Data e Data Encoding . . . . .	12
3.2.4	Analisi esplorativa dei dati e test statistici . . .	13
3.2.5	Stagionalità . . . . .	15

3.2.6	Analisi delle Feature . . . . .	17
3.2.7	Test Statisticci . . . . .	20
3.2.8	Stazionarietà . . . . .	22
<b>4</b>	<b>Forecasting</b>	<b>24</b>
4.0.1	ARIMA . . . . .	24
4.0.2	SARIMA . . . . .	28
4.0.3	Confronto dei Modelli . . . . .	31
<b>5</b>	<b>Deep Learning</b>	<b>33</b>
5.0.1	Multilayer Perceptron . . . . .	34

# Chapter 1

## Notebook Google Colab

### 1.1 Traccia

La traccia del primo progetto dell’elaborato di Information Systems and Business Intelligence prevede di preparare un notebook in Google Colab per il Trattamento dei Dati su un dataset a nostra scelta. In particolare, sono richieste:

- l’inclusione di codice per importare il dataset, eseguire l’analisi esplorativa dei dati, la pulizia e la trasformazione.
- l’applicazione di tecniche di Analisi dei Dati.
- commenti del codice e dei vari passaggi.

## 1.2 Google Colab

Google Colab, abbreviazione di "Google Colaboratory", è una piattaforma di cloud computing offerta gratuitamente da Google. Si tratta di un ambiente di sviluppo basato su browser che consente di scrivere, eseguire e condividere codice Python. Di seguito sono elencati le principali caratteristiche:

- Accesso Gratuito e Basato sul Web
- Ambiente di Esecuzione Basato su Jupyter Notebooks
- Accelerazione GPU e TPU
- Archiviazione su Google Drive
- Integrazione con Google Cloud



Figure 1.1: Logo Google Colab

Google Colab presenta molteplici utilizzi tra cui:

1. Sviluppo e Sperimentazione
2. Formazione e Collaborazione

### 3. Machine Learning e Deep Learning

In sintesi, Google Colab è una piattaforma versatile che facilita lo sviluppo, la collaborazione e l'esecuzione di codice Python, fornendo risorse computazionali accessibili e un ambiente interattivo basato su Jupyter Notebooks.

# Chapter 2

# Dataset

Il dataset utilizzato per questa analisi del cambiamento climatico a Delhi è stato acquisito da Kaggle e può essere trovato al seguente indirizzo: <https://www.kaggle.com/datasets/sumanthvrao/daily-climate-time-series-data>)

## 2.1 Descrizione del Dataset

Il dataset contiene dati sul cambiamento climatico nella città di Delhi, in India, dal 1° gennaio 2013 al 24 aprile 2017. I dati sono rappresentati da 5 indicatori per 1462 giorni.

## 2.2 Descrizione delle features

Il dataset presenta 5 feature per descrivere i 1462 campioni:

- **Data (date):** La data del campionamento dei dati.

- **Temperatura Media (meantemp):** La temperatura media calcolata da intervalli multipli di 3 ore nel corso della giornata.
- **Umidità (humidity):** Il valore di umidità per la giornata, espresso in grammi di vapore acqueo per metro cubo di volume d'aria.
- **Velocità del Vento (wind\_speed):** La velocità del vento misurata in chilometri orari.
- **Pressione Atmosferica (pressure):** La lettura della pressione atmosferica, misurata in atmosfere.

## 2.3 Obiettivo

L'obiettivo principale è comprendere le variazioni climatiche e effettuare previsioni utilizzando competenze di machine learning e analisi dei dati. Questo dataset è stato raccolto per supportare la ricerca nell'ambito delle scienze ambientali, consentendo analisi approfondite e lo sviluppo di modelli predittivi per comprendere meglio il cambiamento climatico nella città di Delhi nel periodo specificato.



# Chapter 3

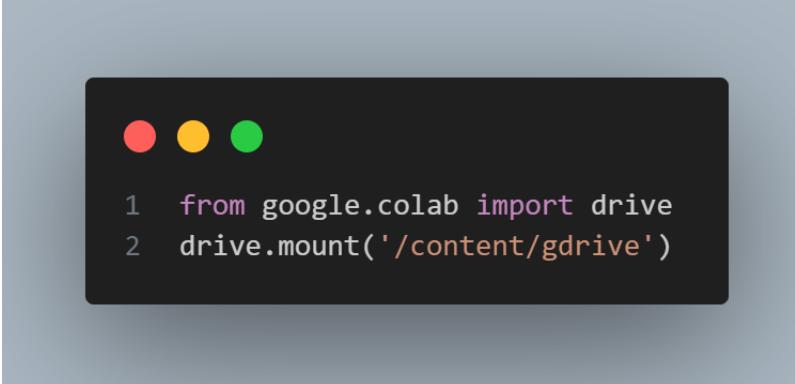
# Analisi e visualizzazione dei dati

## 3.1 Fase Preliminare

Nella presente sezione, esploreremo il codice utilizzato per condurre l’analisi e la visualizzazione dei dati climatici relativi a Delhi. Le operazioni eseguite sono descritte passo dopo passo, insieme alle librerie Python utilizzate.

### 3.1.1 Montaggio di Google Drive

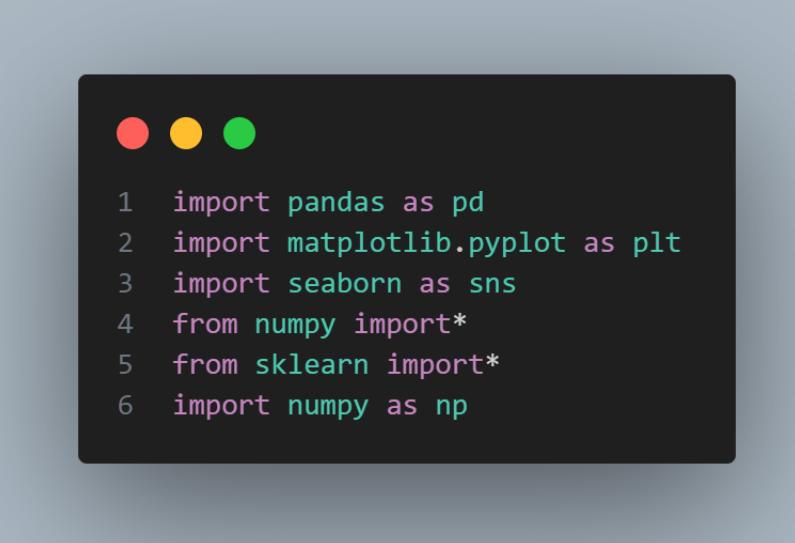
Per garantire l’accesso ai dati locali presenti su Google Drive all’interno dell’ambiente di sviluppo di Google Colab, abbiamo utilizzato il seguente snippet di codice per montare Google Drive:



```
● ● ●  
1 from google.colab import drive  
2 drive.mount('/content/gdrive')
```

### 3.1.2 Importazione Librerie

Successivamente, abbiamo importato diverse librerie Python essenziali per l'analisi e la visualizzazione dei dati, tra cui Pandas, Matplotlib, Seaborn, NumPy e Scikit-Learn:



```
● ● ●  
1 import pandas as pd  
2 import matplotlib.pyplot as plt  
3 import seaborn as sns  
4 from numpy import*  
5 from sklearn import*  
6 import numpy as np
```

Abbiamo introdotto la libreria facets-overview per fornire una panoramica interattiva dei dati. La seguente istruzione è stata eseguita per installare la versione specifica di Facets-Overview:

```
● ● ●  
1 # For facets  
2 from IPython.core.display import display, HTML  
3 import base64  
4 !pip install facets-overview==1.0.0  
5 from facets_overview.feature_statistics_generator import FeatureStatisticsGenerator
```

### 3.1.3 Lettura dei Dati e Statistiche Descrittive

Abbiamo caricato i dati climatici dal file 'DailyDelhiClimateTrain.csv' salvato in Google Drive, utilizzando la libreria Pandas. Successivamente, abbiamo visualizzato alcune statistiche descrittive del dataset attraverso il comando:

```
● ● ●  
1 data = pd.read_csv('/content/gdrive/MyDrive/DailyDelhiClimateTrain.csv')  
2 data.describe()  
3
```

Infine, per migliorare la visualizzazione delle statistiche descrittive, abbiamo utilizzato la funzione **background gradient** di Pandas per applicare una mappa di colore alle celle del dataframe:

```
● ● ●  
1 data.describe().style.background_gradient(cmap='Oranges')
```

I risultati prodotti dai precedenti snippet di codice sono:

	meantemp	humidity	wind_speed	meanpressure
count	1462.000000	1462.000000	1462.000000	1462.000000
mean	25.495521	60.771702	6.802209	1011.104548
std	7.348103	16.769652	4.561602	180.231668
min	6.000000	13.428571	0.000000	-3.041667
25%	18.857143	50.375000	3.475000	1001.580357
50%	27.714286	62.625000	6.221667	1008.563492
75%	31.305804	72.218750	9.238235	1014.944901
max	38.714286	100.000000	42.220000	7679.333333

	meantemp	humidity	wind_speed	meanpressure
count	1462.000000	1462.000000	1462.000000	1462.000000
mean	25.495521	60.771702	6.802209	1011.104548
std	7.348103	16.769652	4.561602	180.231668
min	6.000000	13.428571	0.000000	-3.041667
25%	18.857143	50.375000	3.475000	1001.580357
50%	27.714286	62.625000	6.221667	1008.563492
75%	31.305804	72.218750	9.238235	1014.944901
max	38.714286	100.000000	42.220000	7679.333333

Sono presenti valori anomali sia in termini di valori minimi che massimi in tutte le feature.

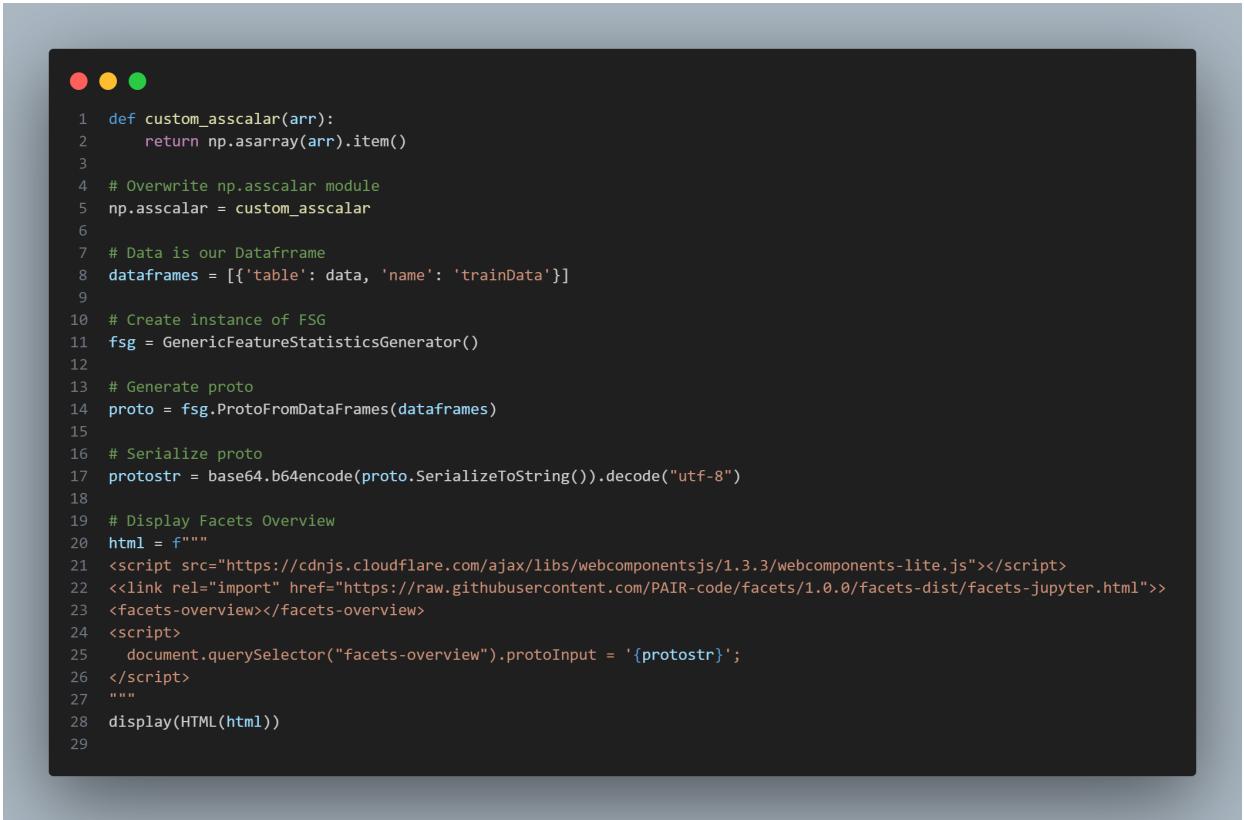
## 3.2 Visualizzazioni dei dati

Andiamo ora ad esplorare i vari modi in cui abbiamo deciso di visualizzare i dati presenti nel Dataset.

### 3.2.1 Visualizzazione dei dati in sfaccettature

Il codice seguente consente di visualizzare statistiche interattive dei dati utilizzando Facets Overview, consentendo un'analisi approfondita

del dataframe fornito (`data`). Questa parte del codice è spesso utilizzata in ambienti come Google Colab o Jupyter Notebook per esplorare e comprendere rapidamente la distribuzione e le caratteristiche dei dati.



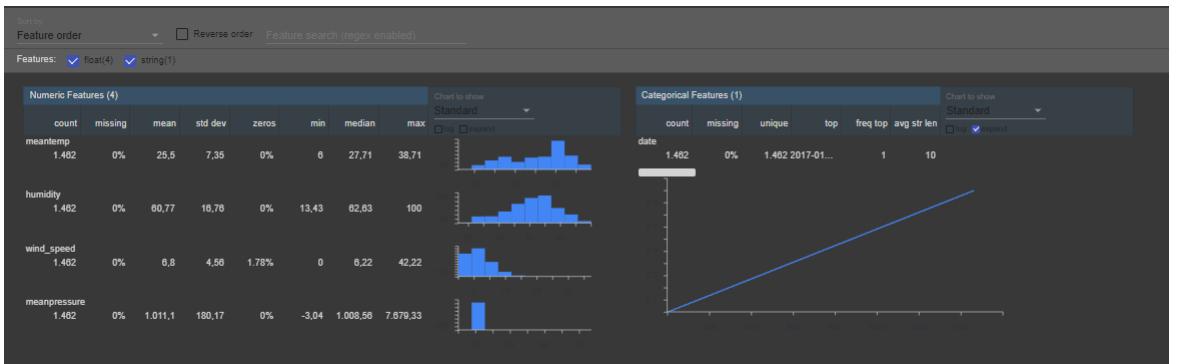
```
● ● ●
1 def custom_asscalar(arr):
2     return np.asarray(arr).item()
3
4 # Overwrite np.asscalar module
5 np.asscalar = custom_asscalar
6
7 # Data is our Dataframe
8 dataframes = [{table: data, name: 'trainData'}]
9
10 # Create instance of FSG
11 fsg = GenericFeatureStatisticsGenerator()
12
13 # Generate proto
14 proto = fsg.ProtoFromDataFrames(dataframes)
15
16 # Serialize proto
17 protostr = base64.b64encode(proto.SerializeToString()).decode("utf-8")
18
19 # Display Facets Overview
20 html = f"""
21 <script src="https://cdnjs.cloudflare.com/ajax/libs/webcomponentsjs/1.3.3/webcomponents-lite.js"></script>
22 <<link rel="import" href="https://raw.githubusercontent.com/PAIR-code/facets/1.0.0/facets-dist/facets-jupyter.html">>
23 <facets-overview></facets-overview>
24 <script>
25   document.querySelector("facets-overview").protoInput = '{protostr}';
26 </script>
27 """
28 display(HTML(html))
29
```

La funzione **custom asscalar** è stata definita per sostituire la funzione **np.asscalar**. La nuova implementazione converte un array in un singolo valore utilizzando **np.asarray(arr).item()**. Questa operazione è stata necessaria in quanto ci dava degli errori con l'implementazione standard della funzione. Il codice sovrascrive il modulo **np.asscalar** con la funzione personalizzata definita in precedenza.

Viene successivamente creato un dizionario contenente un dataframe (**data**) e il suo nome associato (**'trainData'**). Viene creata un'istanza

dell’oggetto **GenericFeatureStatisticsGenerator** dalla libreria Facets Overview ed un protocollo di dati utilizzando la funzione **ProtoFromDataFrames** dell’oggetto **GenericFeatureStatisticsGenerator**. Il protocollo di dati viene serializzato in una stringa codificata **in base64 (protostr)**.

Viene infine costruita una stringa HTML che include gli script necessari per visualizzare Facets Overview. La stringa serializzata (protostr) viene inserita come input per l’anteprima Facets Overview. Infine, la visualizzazione viene mostrata attraverso **display(HTML(html))**. Il risultato di questo snippet è:

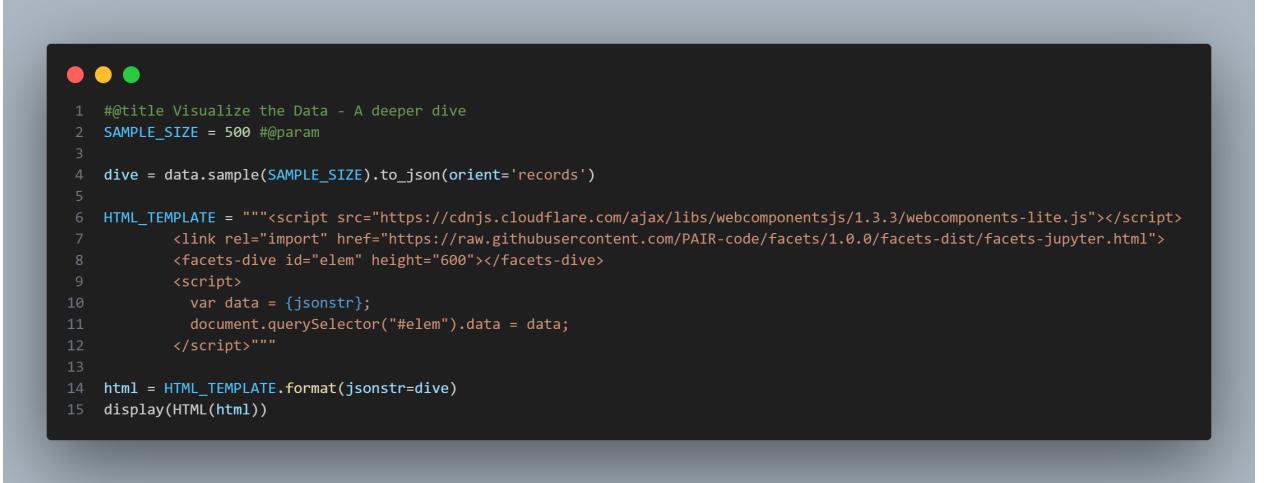


In cui vengono visualizzati i valori mancanti, media, deviazione standard, valori nulli, minimo, massimo e mediana di ogni feature numerica del dataset, e le caratteristiche dell’unica feature categorica del dataset.

### 3.2.2 Una visione più profonda

Il seguente codice consente di esplorare in modo interattivo un campione casuale dei dati utilizzando Facets Dive, che è un tool di visu-

alizzazione utile per comprendere la distribuzione e le relazioni tra le variabili nel nostro dataset.



```
1 #@title Visualize the Data - A deeper dive
2 SAMPLE_SIZE = 500 #@param
3
4 dive = data.sample(SAMPLE_SIZE).to_json(orient='records')
5
6 HTML_TEMPLATE = """<script src="https://cdnjs.cloudflare.com/ajax/libs/webcomponentsjs/1.3.3/webcomponents-lite.js"></script>
7     <link rel="import" href="https://raw.githubusercontent.com/PAIR-code/facets/1.0.0/facets-dist/facets-jupyter.html">
8     <facets-dive id="elem" height="600"></facets-dive>
9     <script>
10         var data = {jsonstr};
11         document.querySelector("#elem").data = data;
12     </script>"""
13
14 html = HTML_TEMPLATE.format(jsonstr=dive)
15 display.HTML(html)
```

Viene definito un parametro **SAMPLE SIZE** che indica la dimensione del campione. In questo caso, il campione è costituito da 500 record estratti casualmente dal dataframe principale. Viene estratto un campione di dimensione **SAMPLE SIZE** dal dataframe principale (data). Questo campione viene convertito in formato JSON utilizzando il metodo **to json** con orientamento '**records**'. L'output (dive) sarà una stringa JSON rappresentante il campione di dati selezionato.

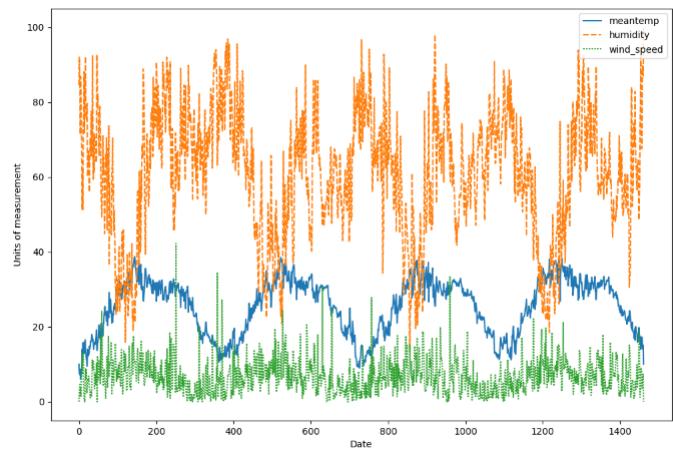
### 3.2.3 Missing Data e Data Encoding

Nonostante il nostro dataset attuale non contenga valori mancanti (Missing Values), abbiamo anticipato la possibilità di future integrazioni di dati o misurazioni aggiuntive che potrebbero contenere valori mancanti. Per affrontare questa eventualità, abbiamo predisposto un codice

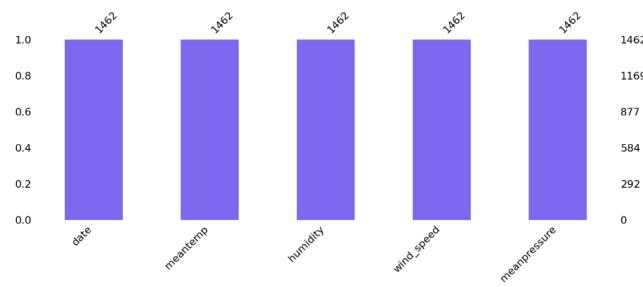
dedicato alla gestione di Missing Values. Preparando preventivamente un codice di gestione dei Missing Values, siamo in grado di garantire la robustezza delle nostre analisi e la coerenza nel trattare eventuali nuovi dati che potrebbero essere integrati nel nostro dataset.

### 3.2.4 Analisi esplorativa dei dati e test statistici

Inizialmente abbiamo provveduto a fare un'analisi dei valori delle singole feature in un'unico grafico lungo il tempo.



Successivamente abbiamo verificato che non ci fossero gap all'interno del dataset



Poi abbiamo analizzato ad uno ad uno i grafici dei valori di ogni singola feature:

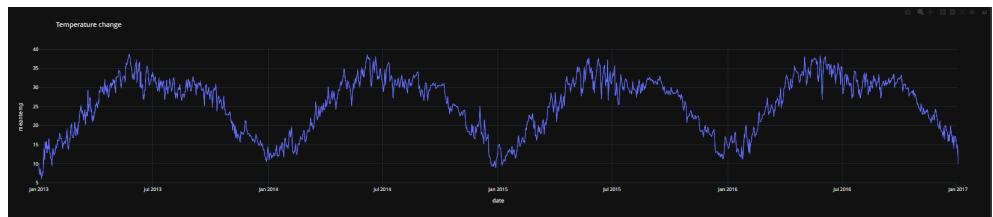


Figure 3.1: Temperatura

Dal grafico della Temperatura possiamo notare che presenta un andamento abbastanza uguale per ogni anno, presentando una temperatura più alta nei mesi estivi ed una temperatura più bassa nei mesi invernali. Deduzione simile per quanto riguarda l’umidità.

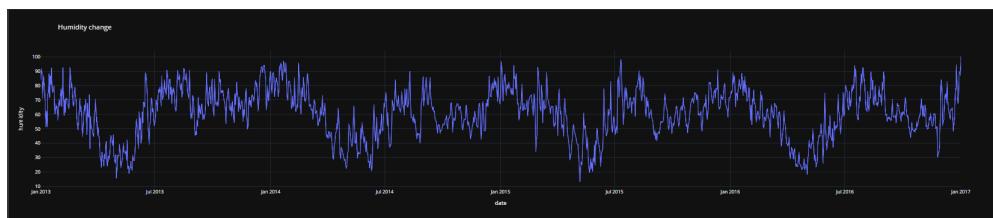


Figure 3.2: Umidità

Il valore del vento rimane uguale, presenta molti valori anomali.

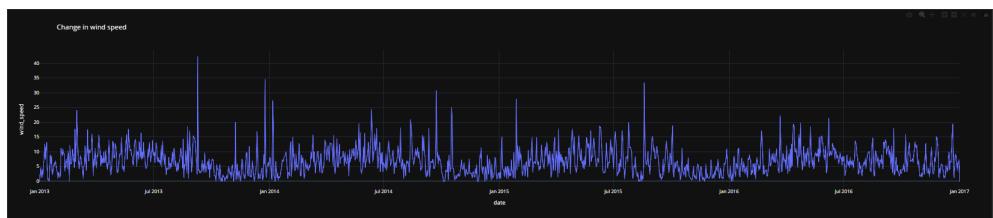


Figure 3.3: Vento

La pressione presenta un valore anomalo nella metà del 2016, il resto rimane abbastanza costante se non per valori diversi nelle misurazioni dell’ultimo anno

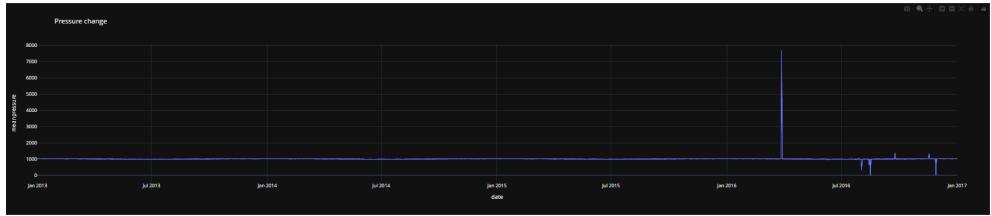


Figure 3.4: Pressione

Il tutto creato tramite dei semplici snippet, di cui mostreremo solo un esempio in quanto tutti uguali (cambia solo la feature da mostrare):



### 3.2.5 Stagionalità

La stagionalità è un aspetto fondamentale nell’analisi del cambiamento climatico, poiché influisce significativamente sui modelli meteorologici e sulle tendenze a lungo termine. In questa, esamineremo la stagionalità nei dati climatici della città di Delhi, India, al fine di comprendere le variazioni stagionali e il loro impatto sul clima locale. La stagionalità si riferisce a variazioni periodiche che si verificano regolarmente durante l’anno. Nel contesto climatico, ciò si traduce in cambiamenti ricorrenti nelle condizioni meteorologiche, che seguono un modello circolare. La stagionalità è composta da tre diversi effetti:

- Tendenza (modello coerente in tutti i dati)
- Stagionale (effetti ciclici)
- Residuo (Errore di previsione)

Riportiamo di seguito i grafici che rappresentano queste tre caratteristiche:

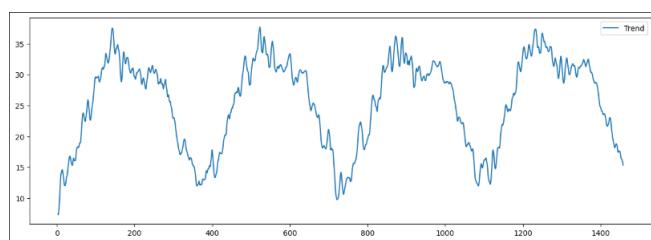


Figure 3.5: Data Trend

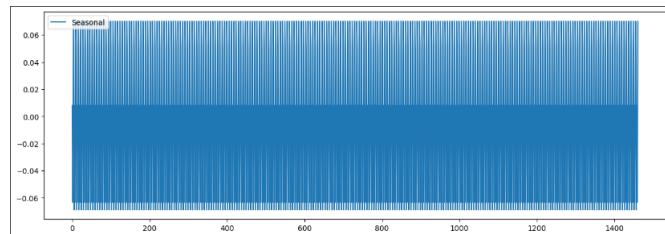


Figure 3.6: Data Seasonal

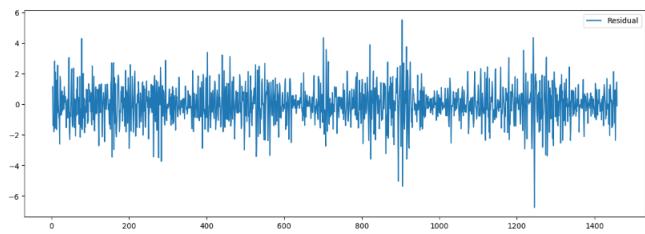


Figure 3.7: Data Residual

Possiamo notare che ci sono grossi residui nei dati

### 3.2.6 Analisi delle Feature

Innanzitutto si sono definite in un array le feature numeriche

```

● ● ●
1 numeric_features=['meantemp', 'humidity', 'wind_speed', 'meanpressure']

```

Andiamo quindi a valutare tracciando il grafico della distibuzione continua delle feature.

```

● ● ●
1 import plotly.graph_objs as go
2 fig = make_subplots(rows=len(numeric_features), cols=3)
3 i=1
4 for feature in numeric_features:
5     fig.add_trace(go.Histogram(x=data[feature], name=feature), row=i, col=1)
6     fig.add_trace(go.Box(x=data[feature], name=feature), row=i, col=2)
7     fig.add_trace(go.Violin(x=data[feature], name=feature), row=i, col=3)
8     i+=1
9 fig.update_layout(height=2400, width=1800, title_text='<b>Continuos distributions', title_x=0.5)
10 fig.show()

```

Un esempio dei risultati di questo snippet è:

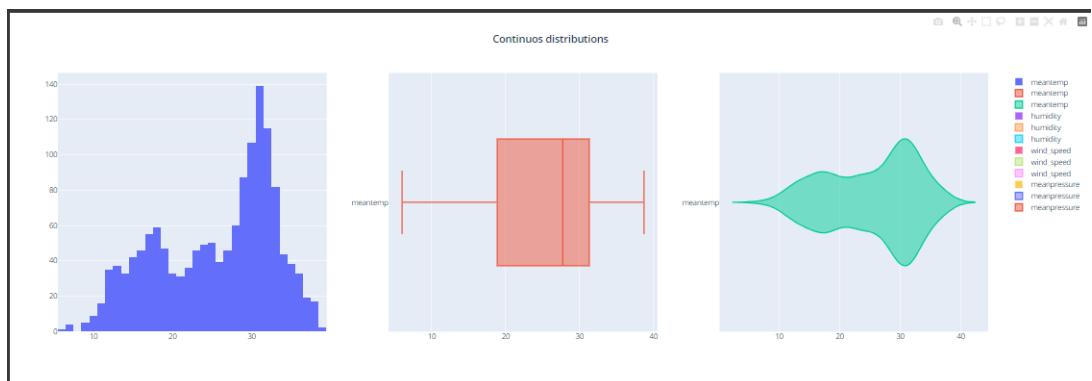
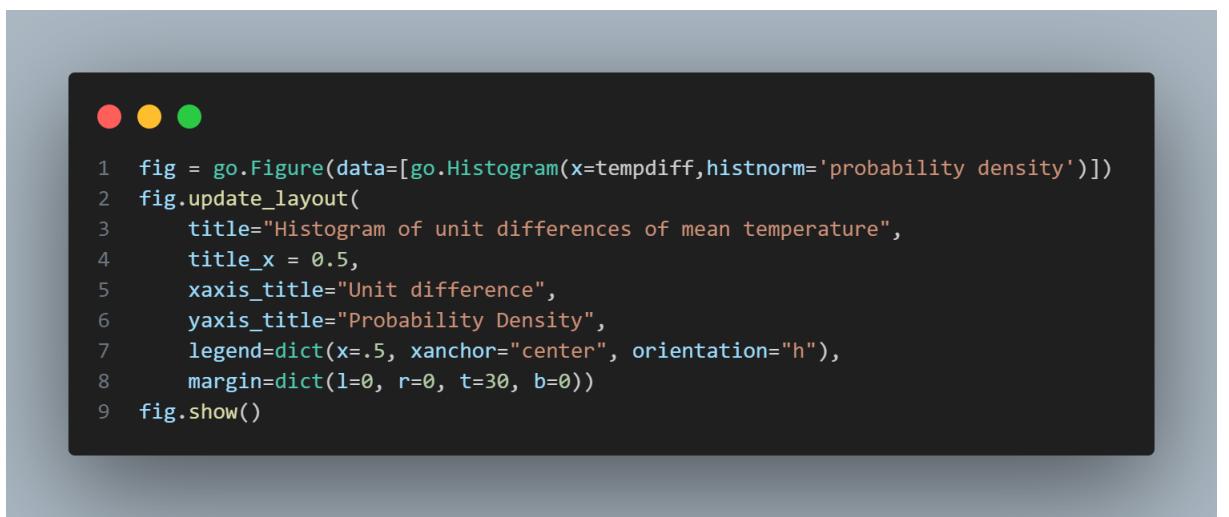


Figure 3.8: Esempio dei grafici di Distribuzione continua della feature relativa alla temperatura media

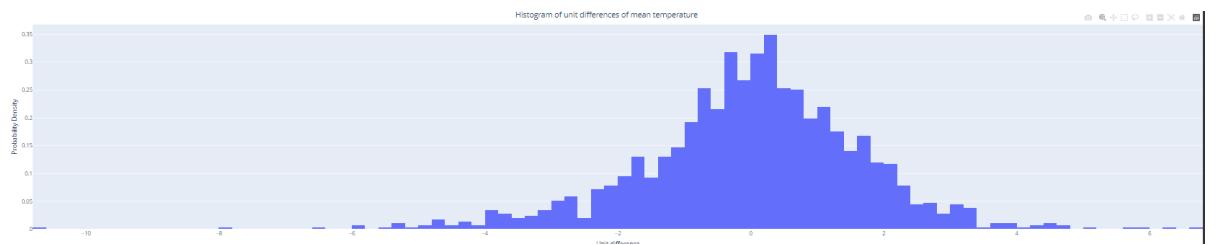
Esistono valori anomali minori sia per temperatura che per l'umidità. Una feature, la pressione media, presenta outliers sia in termini di valori massimi che minimi. Le feature non sono distribuite normalmente, ad eccezione dell'umidità e possibilmente della pressione atmosferica.

Siamo interessati al segno della feature target: la temperatura media. Costruiamo un istogramma della distribuzione delle differenze unitarie per questa feature.

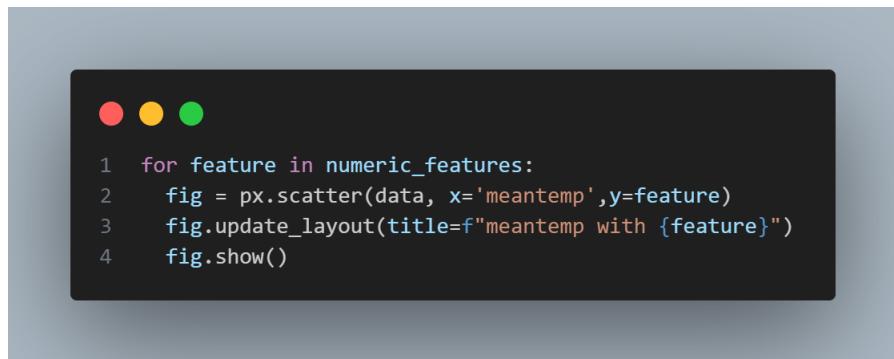


```
● ● ●  
1 fig = go.Figure(data=[go.Histogram(x=tempdiff,histnorm='probability density')])  
2 fig.update_layout(  
3     title="Histogram of unit differences of mean temperature",  
4     title_x = 0.5,  
5     xaxis_title="Unit difference",  
6     yaxis_title="Probability Density",  
7     legend=dict(x=.5, xanchor="center", orientation="h"),  
8     margin=dict(l=0, r=0, t=30, b=0))  
9 fig.show()
```

Il risultato di questo snippet è un istogramma:



Costruiamo grafici a dispersione della nostra feature target con altre feature nella tabella tramite questo snippet di codice.



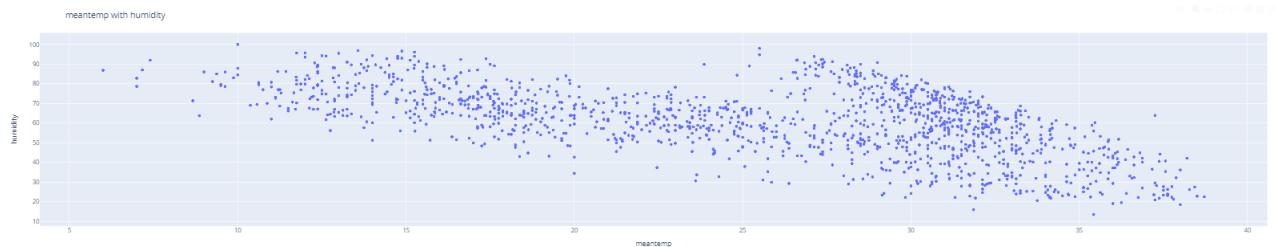
```

● ● ●

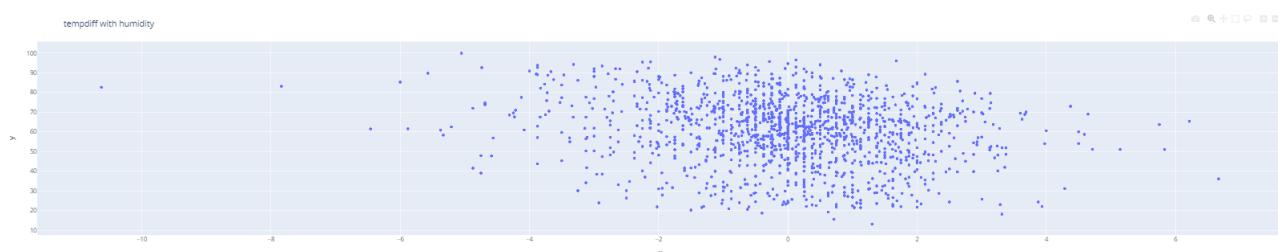
1 for feature in numeric_features:
2     fig = px.scatter(data, x='meantemp',y=feature)
3     fig.update_layout(title=f"meantemp with {feature}")
4     fig.show()

```

Di maggiore rilevanza è la relazione tra umidità e temperatura media:

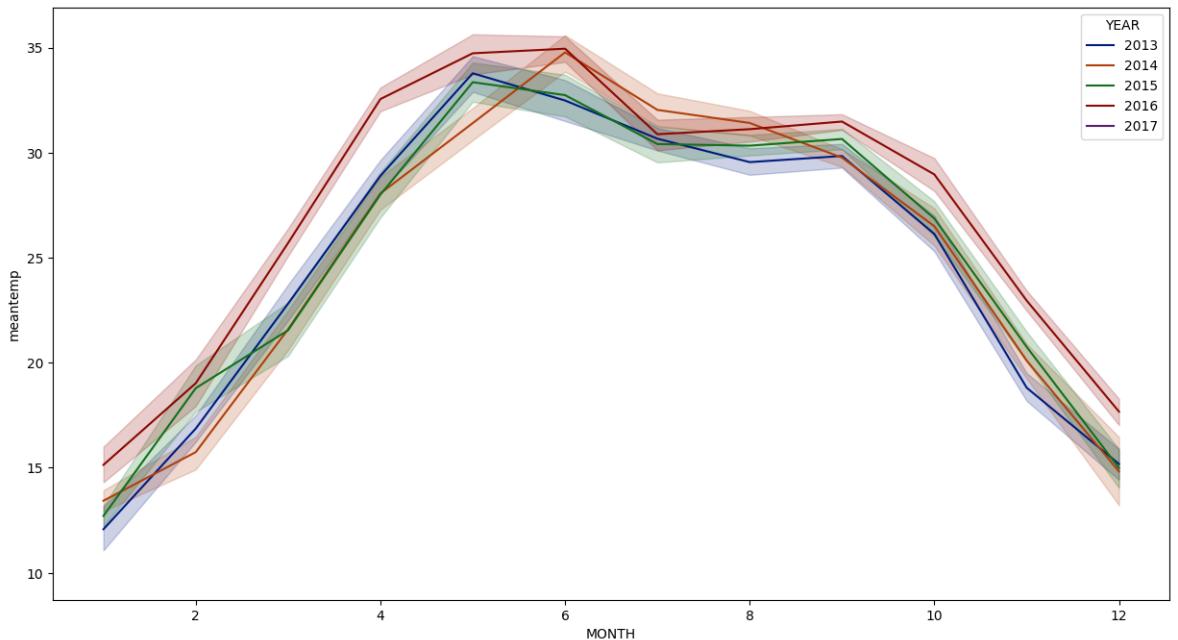


Esiste una leggera correlazione inversa con il segno dell'umidità.  
Approfondiamo il loro rapporto:



Dal grafico qui sopra si evince che non è possibile prevedere il valore della temperatura dal valore dell'umidità del giorno precedente.

Infine abbiamo voluto valutare come la temperatura cambiasse negli stessi periodi anno per anno.



Come possiamo vedere, la temperatura media aumenta di anno in anno. Ciò è dovuto principalmente al cambiamento climatico globale.

### 3.2.7 Test Statistici

Eseguiamo test statistici con il nostro dataset. Si propone di iniziare con la costruzione di una matrice di correlazione. Tramite due semplici Snippet possiamo andare a creare una matrice di correlazione.

```
● ● ●  
1 correlation = data.corr()  
2 print(correlation['meantemp'].sort_values(ascending = False), '\n')
```

Dopo aver calcolato la correlazione, plottiamo la matrice.

```

● ● ●
1 k= 10
2 cols = correlation.nlargest(k,'meantemp')['meantemp'].index
3 print(cols)
4 cm = np.corrcoef(data[cols].values.T)
5 mask = np.triu(np.ones_like(data.corr()))
6 f , ax = plt.subplots(figsize = (14,12))
7 sns.heatmap(cm,mask=mask, vmax=.8, linewidths=0.01,square=True,annot=True,cmap='viridis',
8             linecolor="white",xticklabels = cols.values ,annot_kws = {'size':12},yticklabels = cols.values)

```

Il risultato di questi snippet è:

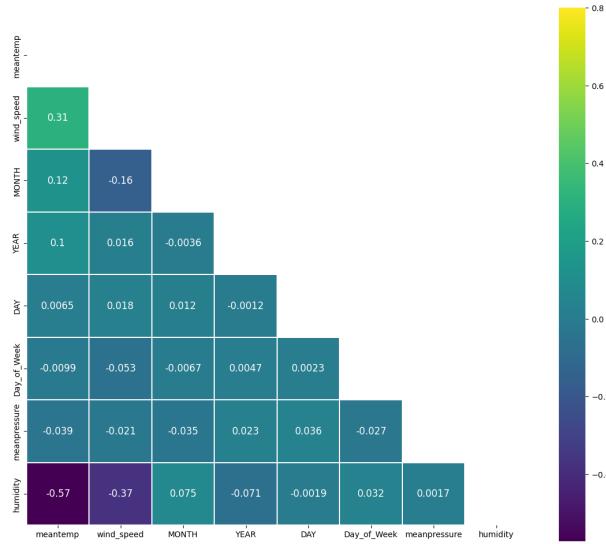


Figure 3.9: Matrice di Correlazione

Notiamo che non c'è correlazione tra umidità e temperatura.

N.B la feature date è stata scomposta in YEAR,MONTH, DAY.

Controlliamo se c'è significatività statistica tra anno, mese, giorno e temperatura media tramite il Mann-Whitney Test e si è notato che c'è una significatività statistica nei dati.

### 3.2.8 Stazionarietà

Controlliamo la stazionarietà della nostra feature target. Abbiamo creato una funzione per la creazione del grafico per effettuare la Time Series Analysis per studiare la stazionarietà. La stazionarietà di un grafico si riferisce alla costanza statistica delle proprietà del grafico nel tempo. In un contesto temporale, un processo o una serie temporale si dice stazionario quando le sue proprietà statistiche, come la media, la varianza e l'autocorrelazione, non cambiano significativamente nel tempo.



```
1  ef tsplot(y, lags=None, figsize=(12, 7), style='bmh'):
2      if not isinstance(y, pd.Series):
3          y = pd.Series(y)
4      with plt.style.context(style):
5          fig = plt.figure(figsize=figsize)
6          layout = (2, 2)
7          ts_ax = plt.subplot2grid(layout, (0, 0), colspan=2)
8          acf_ax = plt.subplot2grid(layout, (1, 0))
9          pacf_ax = plt.subplot2grid(layout, (1, 1))
10
11         y.plot(ax=ts_ax)
12         ts_ax.set_title('Time series analysis')
13         smt.graphics.plot_acf(y, lags=lags, ax=acf_ax, alpha=0.5)
14         smt.graphics.plot_pacf(y, lags=lags, ax=pacf_ax, alpha=0.5)
15
16         print("Dickey-Fuller criterion: p=%f" % sm.tsa.stattools.adfuller(y)[1])
17
18         plt.tight_layout()
19     return
20
21 tsplot(data.meantemp, lags=30)
```

Figure 3.10: Snippet per la creazione del grafico

La Serie non risulta essere stazionaria.

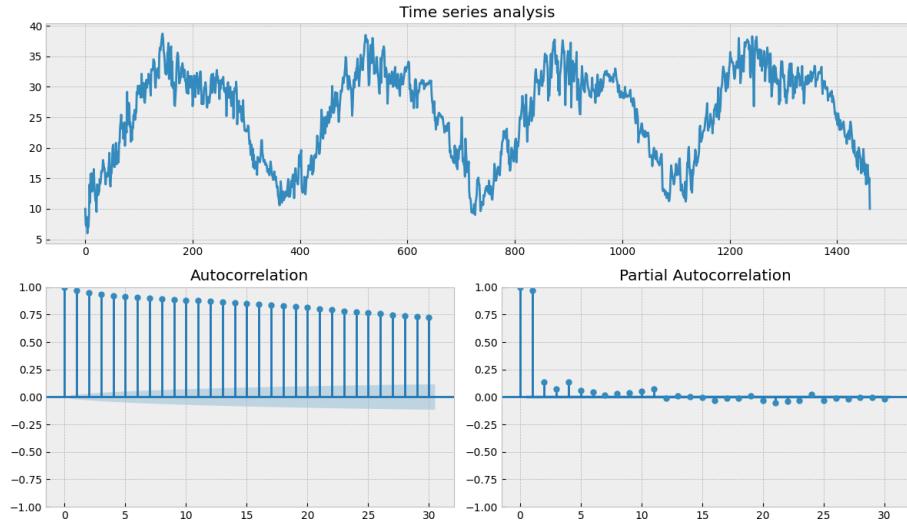


Figure 3.11: Stazionarietà

Dai grafici possiamo dedurre che il test Dickey-Fuller, un test statistico utilizzato per verificare la presenza di radici unitarie in una serie temporale, il che è essenziale per determinare se la serie è stazionaria, è uguale a zero, ma presenta un'elevata varianza nella forma del grafico.

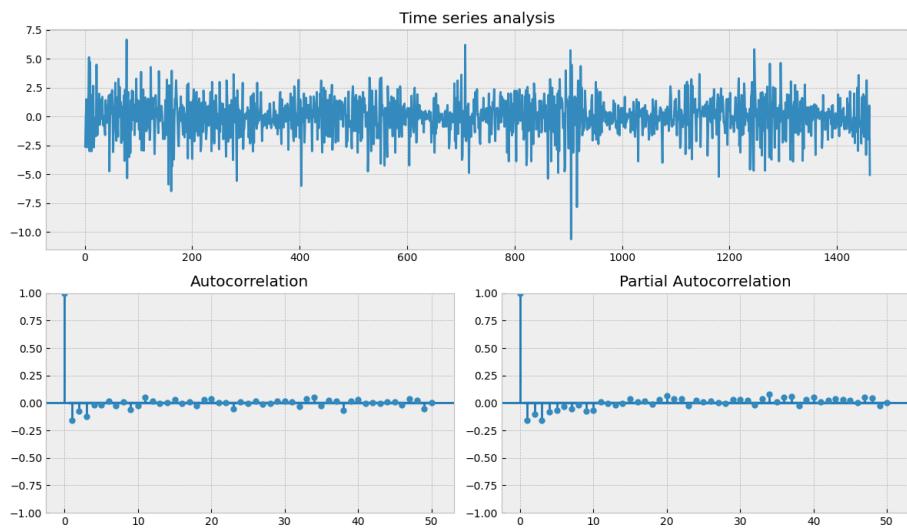


Figure 3.12: Dickey-Fuller Test

# Chapter 4

## Forecasting

Il forecasting, o previsione, è il processo di stima o anticipazione di eventi futuri basato su dati storici e analisi delle tendenze. In altre parole, è un tentativo di fare previsioni o proiezioni sulla base di dati passati, al fine di ottenere una visione più chiara del futuro e prendere decisioni informate. Nel contesto aziendale, il forecasting è spesso utilizzato per prevedere le vendite, la domanda di prodotti, i flussi finanziari, la produzione, e altre variabili rilevanti per la gestione dell'azienda. Esso si basa su dati storici e può coinvolgere l'uso di modelli statistici, algoritmi di machine learning, o metodi analitici più tradizionali per sviluppare previsioni.

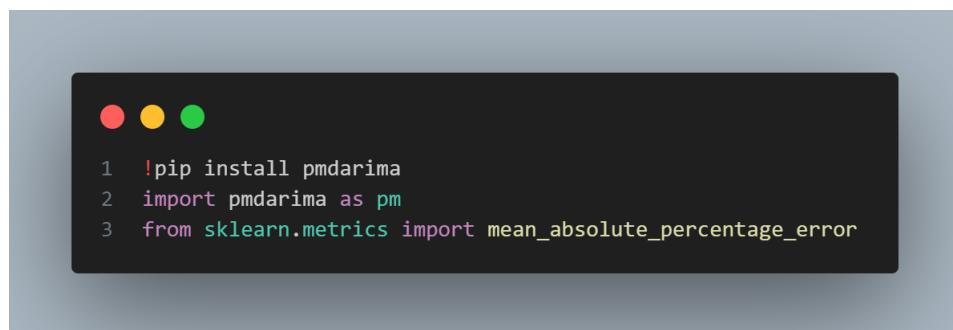
### 4.0.1 ARIMA

Questa sezione si concentra sulla previsione di serie temporali utilizzando modelli ARIMA («Autoregressive Integrated Moving Average»),

da applicare ai dati storici per ottenere previsioni, ampiamente utilizzato in previsioni finanziarie, analisi delle vendite, previsioni di traffico, previsioni meteorologiche e in molte altre aree in cui è importante comprendere e prevedere pattern temporali.

- **AR - Autoregressive:** questo componente considera la relazione tra l'osservazione attuale e le sue precedenti. L'ordine dell'autoregressione, denotato come ' $p$ ', indica quanti periodi temporali passati vengono considerati.
- **MA - Moving Average:** questo elemento tiene conto della relazione tra l'errore residuo attuale e i suoi valori precedenti in una finestra mobile. L'ordine della media mobile, denotato come ' $q$ ', specifica il numero di errori residui precedenti utilizzati nel modello.
- **I - Integrated:** questo rappresenta la parte "integrata" del modello e riguarda il grado di differenziazione applicato alla serie temporale per renderla stazionaria. L'ordine di differenziazione, denotato come ' $d$ ', indica quante volte è necessario differenziare la serie temporale per renderla stazionaria.

Innanzitutto abbiamo installato la libreria che contiene i modelli ARIMA su Colab:



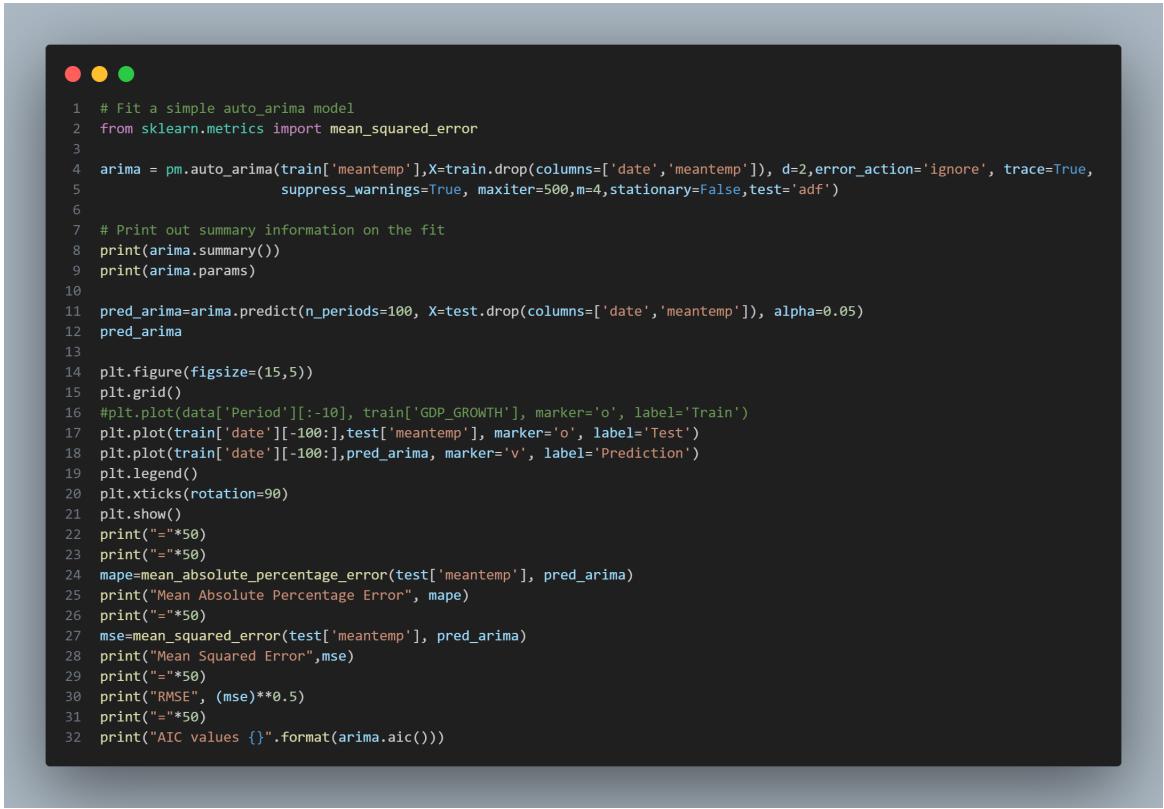
```
1 !pip install pmdarima
2 import pmdarima as pm
3 from sklearn.metrics import mean_absolute_percentage_error
```

Successivamente abbiamo definito il Training Set e lo abbiamo splittato in Training e Test



```
1 train = pd.read_csv('/content/gdrive/MyDrive/DailyDelhiClimateTrain.csv')
2
3 train=train[:-100]
4 test=train[-100:]
5
6 plt.figure(figsize=(35,5))
7 plt.grid()
8 plt.plot(train['meantemp'], marker='v', label='Train')
9 plt.plot( test['meantemp'], marker='o', label='Test')
10 plt.xticks(rotation=90)
11 plt.legend()
12 plt.show()
```

Infine abbiamo definito il modello.



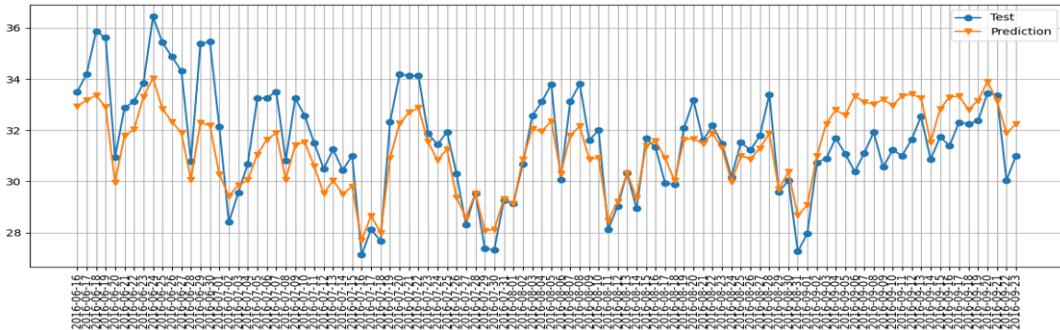
```

1 # Fit a simple auto_arima model
2 from sklearn.metrics import mean_squared_error
3
4 arima = pm.auto_arima(train['meantemp'],X=train.drop(columns=['date','meantemp']), d=2,error_action='ignore', trace=True,
5 suppress_warnings=True, maxiter=500,m=4,stationary=False,test='adf')
6
7 # Print out summary information on the fit
8 print(arima.summary())
9 print(arima.params)
10
11 pred_arima=arima.predict(n_periods=100, X=test.drop(columns=['date','meantemp']), alpha=0.05)
12 pred_arima
13
14 plt.figure(figsize=(15,5))
15 plt.grid()
16 #plt.plot(data['Period'][:-10], train['GDP_GROWTH'], marker='o', label='Train')
17 plt.plot(train['date'][-100:],test['meantemp'], marker='o', label='Test')
18 plt.plot(train['date'][-100:],pred_arima, marker='v', label='Prediction')
19 plt.legend()
20 plt.xticks(rotation=90)
21 plt.show()
22 print("*"*50)
23 print("*"*50)
24 mape=mean_absolute_percentage_error(test['meantemp'], pred_arima)
25 print("Mean Absolute Percentage Error", mape)
26 print("*"*50)
27 mse=mean_squared_error(test['meantemp'], pred_arima)
28 print("Mean Squared Error",mse)
29 print("*"*50)
30 print("RMSE", (mse)**0.5)
31 print("*"*50)
32 print("AIC values {}".format(arima.aic()))

```

I coefficienti rappresentano gli effetti stimati di diverse variabili sul modello:

- L'umidità ha un coefficiente di -0,1389, indicando una relazione negativa con la variabile dipendente.
- La velocità del vento ha un coefficiente di -0,0274, mostrando anch'esso una relazione negativa.
- Il coefficiente per la pressione media sembra essere statisticamente non significativo poiché il suo valore p (0,935) è molto più alto del livello di significatività comunemente usato, che è di 0,05.



I valori indicano una buona precisione delle previsioni del modello ARIMA, con un MAPE, MSE e RMSE relativamente bassi, suggerendo che le previsioni sono in linea con i valori reali delle serie temporali. L'AIC, con un valore più basso, suggerisce che il modello ha una buona adattabilità ai dati senza essere troppo complesso.

```
=====
Mean Absolute Percentage Error 0.03295482930571065
=====
Mean Squared Error 1.7413113101789426
=====
RMSE 1.3195875530554775
=====
AIC values 4690.464380994352
```

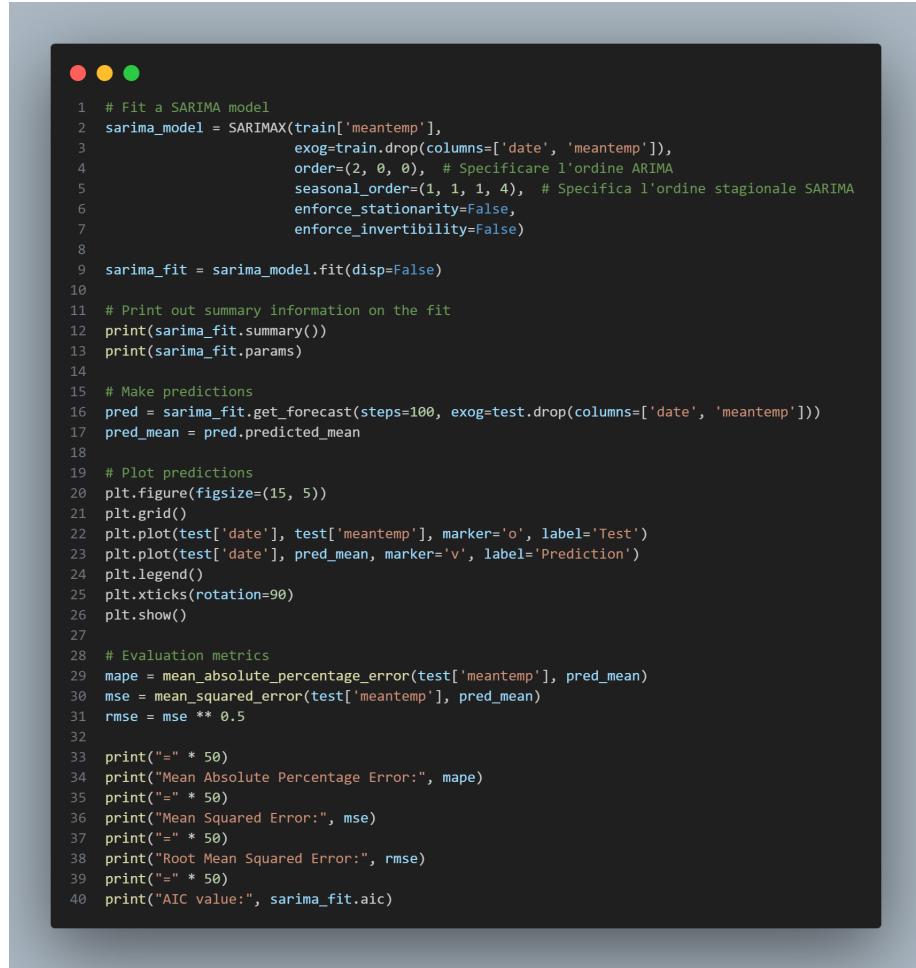
## 4.0.2 SARIMA

SARIMA, acronimo di «Seasonal Autoregressive Integrated Moving Average», è un'estensione del modello ARIMA progettata per gestire la stagionalità presente nei dati di serie temporali. Gli ordini stagionali di SARIMA sono denotati da  $(p, d, q, s)$ , dove:  $p$ = ordine dell'autoregressione stagionale,  $d$ = ordine di differenziazione stagionale,  $q$ = ordine della media mobile stagionale,  $s$ = periodo della stagional-

ità. SARIMA modella non solo le relazioni tra i valori attuali e i valori precedenti, ma considera anche la relazione tra i valori stagionali attuali e quelli dei periodi stagionali precedenti.

Le Operazioni preliminari sono state le stesse per ARIMA.

Definiamo successivamente il nostro modello:



```

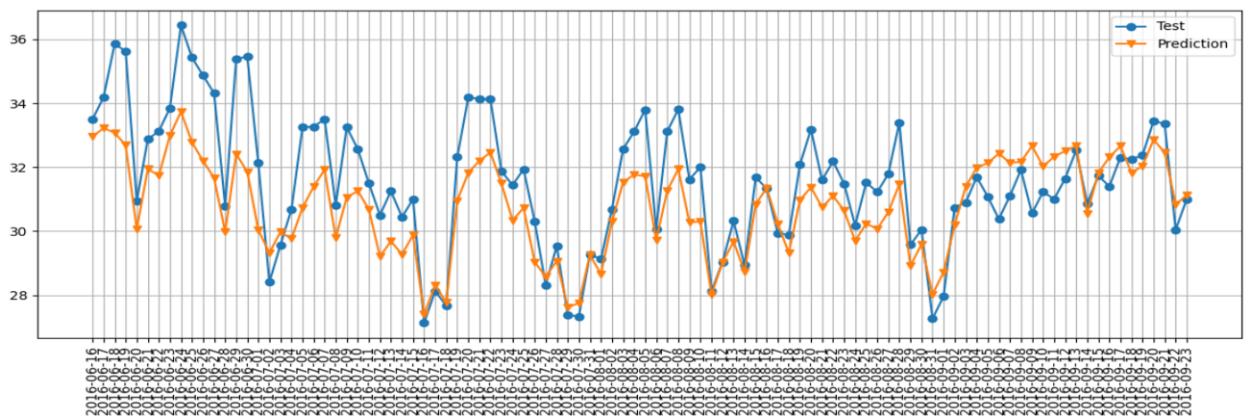
1 # Fit a SARIMA model
2 sarima_model = SARIMAX(train['meantemp'],
3                         exog=train.drop(columns=['date', 'meantemp']),
4                         order=(2, 0, 0), # Specificare l'ordine ARIMA
5                         seasonal_order=(1, 1, 1, 4), # Specifica l'ordine stagionale SARIMA
6                         enforce_stationarity=False,
7                         enforce_invertibility=False)
8
9 sarima_fit = sarima_model.fit(disp=False)
10
11 # Print out summary information on the fit
12 print(sarima_fit.summary())
13 print(sarima_fit.params)
14
15 # Make predictions
16 pred = sarima_fit.get_forecast(steps=100, exog=test.drop(columns=['date', 'meantemp']))
17 pred_mean = pred.predicted_mean
18
19 # Plot predictions
20 plt.figure(figsize=(15, 5))
21 plt.grid()
22 plt.plot(test['date'], test['meantemp'], marker='o', label='Test')
23 plt.plot(test['date'], pred_mean, marker='v', label='Prediction')
24 plt.legend()
25 plt.xticks(rotation=90)
26 plt.show()
27
28 # Evaluation metrics
29 mape = mean_absolute_percentage_error(test['meantemp'], pred_mean)
30 mse = mean_squared_error(test['meantemp'], pred_mean)
31 rmse = mse ** 0.5
32
33 print("=" * 50)
34 print("Mean Absolute Percentage Error:", mape)
35 print("=" * 50)
36 print("Mean Squared Error:", mse)
37 print("=" * 50)
38 print("Root Mean Squared Error:", rmse)
39 print("=" * 50)
40 print("AIC value:", sarima_fit.aic)

```

I coefficienti rappresentano gli effetti stimati di diverse variabili sul modello:

- L'umidità ha un coefficiente di -0,1340 con un errore standard di 0,003, anch'esso con un valore p molto basso.

- La velocità del vento ha un coefficiente di -0,0197 con un errore standard di 0,007 e un valore p che indica significatività statistica, ma meno marcata rispetto al primo output.
  - La pressione media, ancora una volta, mostra un valore p molto alto di 0,901, indicando una mancanza di significatività statistica.

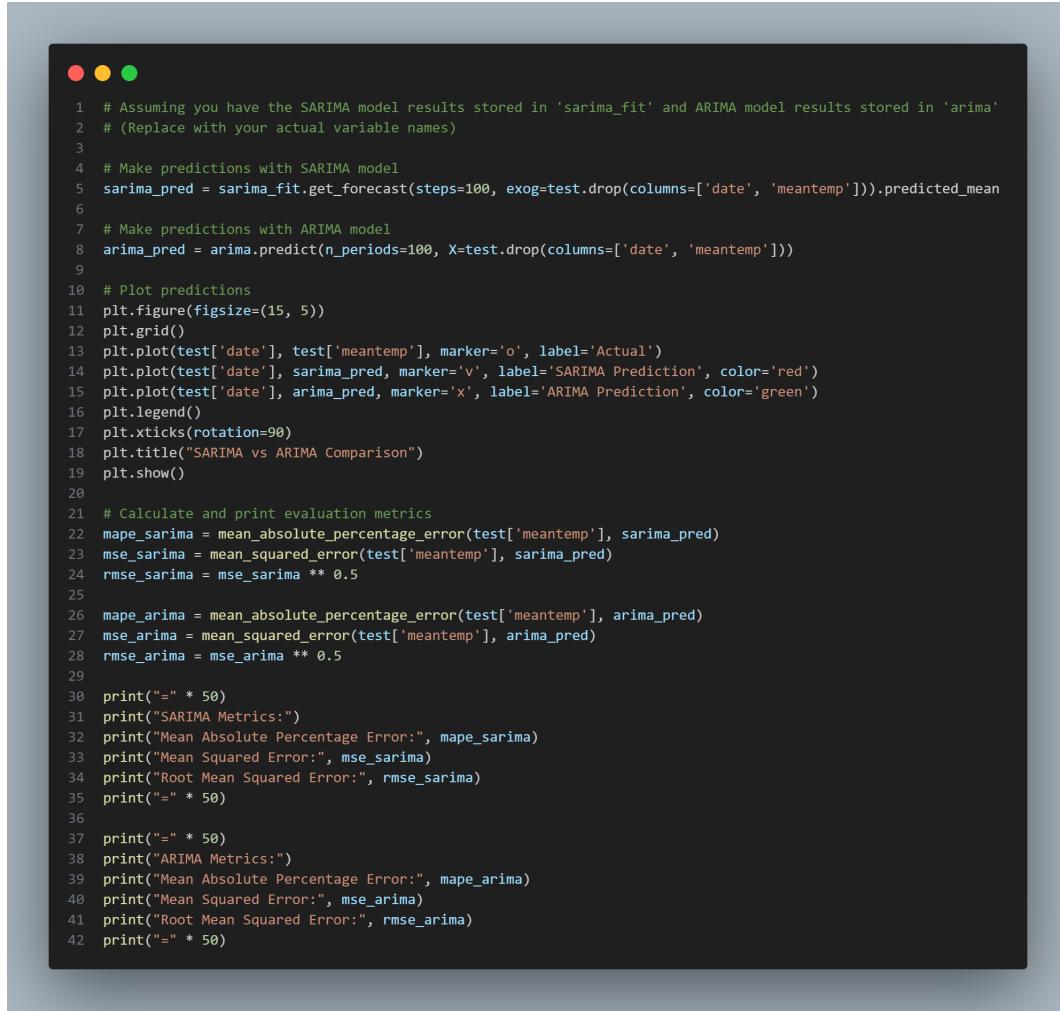


Risultati comparabili al modello precedente, che indicano una buona precisione delle previsioni del modello SARIMA (MAPE, MSE e RMSE relativamente bassi suggeriscono previsioni vicine ai valori reali delle serie temporali, e l'AIC basso suggerisce che il modello ha un buon adattamento ai dati senza essere troppo complesso).

```
-----  
Mean Absolute Percentage Error: 0.03342730015454057  
-----  
Mean Squared Error: 1.8147794705471707  
-----  
Root Mean Squared Error: 1.3471375098879739  
-----  
AIC value: 4536.650150737484
```

### 4.0.3 Confronto dei Modelli

Abbiamo costruito dei grafici per il confronto tra i due modelli per stabilirne uno migliore o valutarne le differenze di prestazioni.



```

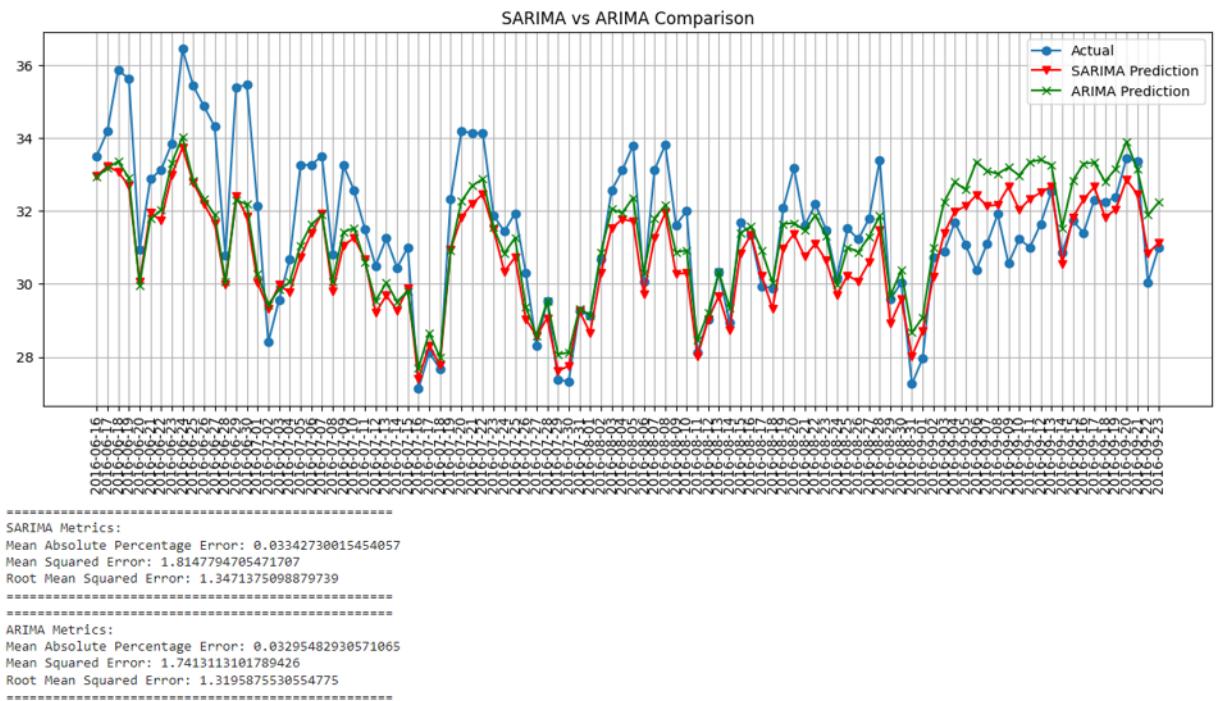
1 # Assuming you have the SARIMA model results stored in 'sarima_fit' and ARIMA model results stored in 'arima'
2 # (Replace with your actual variable names)
3
4 # Make predictions with SARIMA model
5 sarima_pred = sarima_fit.get_forecast(steps=100, exog=test.drop(columns=['date', 'meantemp'])).predicted_mean
6
7 # Make predictions with ARIMA model
8 arima_pred = arima.predict(n_periods=100, X=test.drop(columns=['date', 'meantemp']))
9
10 # Plot predictions
11 plt.figure(figsize=(15, 5))
12 plt.grid()
13 plt.plot(test['date'], test['meantemp'], marker='o', label='Actual')
14 plt.plot(test['date'], sarima_pred, marker='v', label='SARIMA Prediction', color='red')
15 plt.plot(test['date'], arima_pred, marker='x', label='ARIMA Prediction', color='green')
16 plt.legend()
17 plt.xticks(rotation=90)
18 plt.title("SARIMA vs ARIMA Comparison")
19 plt.show()
20
21 # Calculate and print evaluation metrics
22 mape_sarima = mean_absolute_percentage_error(test['meantemp'], sarima_pred)
23 mse_sarima = mean_squared_error(test['meantemp'], sarima_pred)
24 rmse_sarima = mse_sarima ** 0.5
25
26 mape_arima = mean_absolute_percentage_error(test['meantemp'], arima_pred)
27 mse_arima = mean_squared_error(test['meantemp'], arima_pred)
28 rmse_arima = mse_arima ** 0.5
29
30 print("=" * 50)
31 print("SARIMA Metrics:")
32 print("Mean Absolute Percentage Error:", mape_sarima)
33 print("Mean Squared Error:", mse_sarima)
34 print("Root Mean Squared Error:", rmse_sarima)
35 print("=" * 50)
36
37 print("=" * 50)
38 print("ARIMA Metrics:")
39 print("Mean Absolute Percentage Error:", mape_arima)
40 print("Mean Squared Error:", mse_arima)
41 print("Root Mean Squared Error:", rmse_arima)
42 print("=" * 50)

```

I modelli hanno prestazioni molto simili nelle previsioni delle serie temporali, con una seppur leggera superiorità di ARIMA nelle metriche di errore che potrebbe indicare una migliore adattabilità dei dati o una maggiore precisione delle previsioni per il dataset specifico in esame.

## CHAPTER 4. FORECASTING

---



# Chapter 5

## Deep Learning

Il Deep Learning è una branca avanzata del machine learning che utilizza reti neurali artificiali profonde per apprendere e rappresentare pattern complessi nei dati. A differenza dei modelli di machine learning tradizionali, il Deep Learning impiega architetture neurali profonde con molteplici strati (noti come reti neurali profonde o DNN) per estrarre automaticamente caratteristiche gerarchiche dai dati.

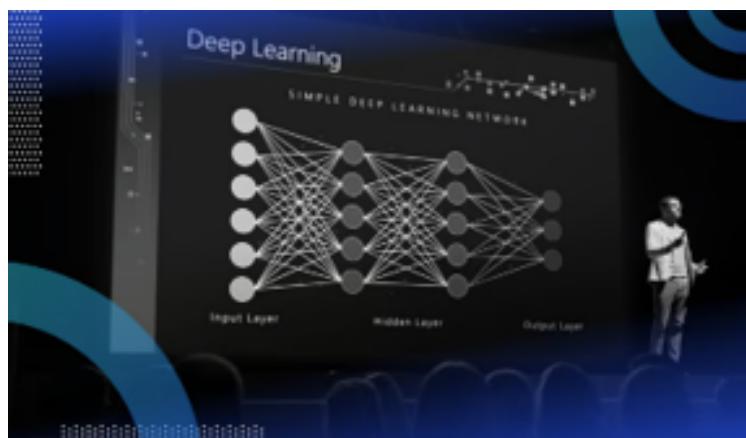
Uno dei tipi più comuni di reti neurali utilizzate nel Deep Learning è il Multilayer Perceptron (MLP), ma ci sono anche altre architetture avanzate come le reti neurali convoluzionali (CNN) per il riconoscimento di immagini e le reti neurali ricorrenti (RNN) per il trattamento di dati sequenziali.

Il successo del Deep Learning è stato alimentato dalla disponibilità di grandi set di dati e dalla potenza di calcolo delle moderne GPU. L'apprendimento profondo ha dimostrato eccellenti risultati in

una vasta gamma di applicazioni, inclusi il riconoscimento di immagini, la traduzione automatica, l’elaborazione del linguaggio naturale, il riconoscimento vocale e molte altre.

La principale caratteristica distintiva del Deep Learning è la sua capacità di apprendere rappresentazioni automatiche e di adattarsi a compiti complessi, riducendo la necessità di estrazione manuale di features. Ciò lo rende particolarmente adatto per problemi in cui la complessità delle relazioni nei dati è elevata.

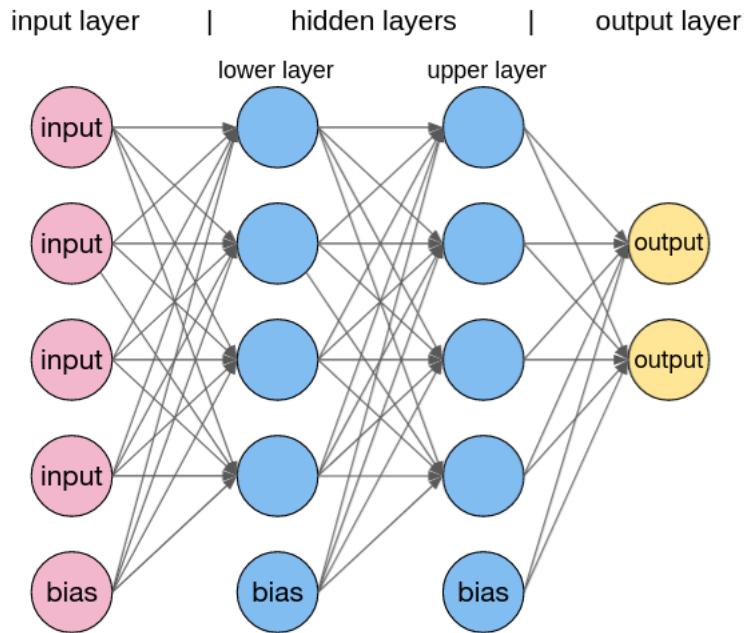
Il Deep Learning sta rivoluzionando molte industrie, offrendo soluzioni avanzate per problemi precedentemente considerati difficili o impossibili da risolvere con metodi tradizionali di machine learning.



### 5.0.1 Multilayer Perceptron

Abbiamo scelto di utilizzare MLP (Multilayer Perceptron) come modello perché per la previsione delle serie temporali presenta diversi vantaggi: acquisizione di relazioni temporali, flessibilità nell’ingegnerizzazione delle caratteristiche, gestione di relazioni non lineari.

1 Multilayer Perceptron (MLP) è un tipo di rete neurale artificiale che rappresenta una classe di modelli di machine learning. Si basa su un'architettura a strati, composta da uno strato di input, uno o più strati nascosti e uno strato di output. Ogni strato è composto da nodi, noti come neuroni, che sono interconnessi attraverso pesi. L'MLP è in grado di apprendere relazioni complesse nei dati grazie al processo di apprendimento supervisionato. Durante la fase di addestramento, gli input vengono presentati alla rete e le sue previsioni vengono confrontate con i valori desiderati. L'algoritmo di retropropagazione dell'errore viene utilizzato per aggiornare i pesi della rete, migliorando progressivamente le sue prestazioni. Caratteristiche chiave dell'MLP includono la capacità di modellare relazioni non lineari, l'adattabilità a una vasta gamma di problemi di machine learning, e la gestione di input di dimensioni variabili. L'MLP è ampiamente impiegato in diversi campi, tra cui riconoscimento di immagini, elaborazione del linguaggio naturale, previsione e classificazione. La sua versatilità lo rende uno strumento potente per la risoluzione di problemi complessi attraverso l'apprendimento automatico.



Abbiamo costruito il modello in questo modo

```

 1 # Preprocessing and Creating Lagged Features
 2 def create_lagged_features(data, n_lags=10):
 3     lagged_df = data.copy()
 4     for i in range(1, n_lags + 1):
 5         lagged_df[f'lag_{i}'] = lagged_df['meantemp'].shift(i)
 6     lagged_df = lagged_df.dropna()
 7     return lagged_df
 8
 9 n_lags = 5 # Number of lags (can be tuned)
10 lagged_df = create_lagged_features(data, n_lags)
11
12 # Splitting Data
13 X = lagged_df[[f'lag_{i}' for i in range(1, n_lags + 1)]]
14 y = lagged_df['meantemp']
15 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=False)
16
17 # Scaling Features
18 scaler = StandardScaler()
19 X_train_scaled = scaler.fit_transform(X_train)
20 X_test_scaled = scaler.transform(X_test)
21
22 # Define MLP Model
23 mlp = MLPRegressor(hidden_layer_sizes=(5, 30), activation='relu', max_iter=1000)
24
25 # Train the Model
26 mlp.fit(X_train_scaled, y_train)
27 # Evaluate the Model
28 y_pred = mlp.predict(X_test_scaled)
29 errors = y_test - y_pred
30 mse = mean_squared_error(y_test, y_pred)
31 print(f'Mean Squared Error: {mse}')

```

Questa è la valutazione del MLP sul nostro dataset.

