In [1]:
```
# 1) Setup: librerie, connessione e lettura dati

import duckdb
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error

con = duckdb.connect(database=':memory:') # connessione in-memory a un database DuckDB
```

In [2]:
```
# caricamento del CSV del dataset in una tabella DuckDB
con.sql("""
    CREATE TABLE crashes_data AS
    SELECT * FROM read_csv_auto('Kaggle_traffic_accidents_dataset.csv')
""")
```

In [3]:
```
# visualizzazione dei primi record come DataFrame Pandas
df = con.sql("SELECT * FROM crashes_data").df()
df.head() # mostra le prime 5 righe
```

Out[3]:

	CRASH_RECORD_ID	CRASH_DATE_EST_I	CRASH_DATE	POSTED_SPEED_LIMIT	TRAFFIC_CONTROL_DEVICE	DEVICE_CONDITION
0	23a79931ef555d54118f64dc9be2cf2dbf59636ce253f7...	None	2023-09-05 19:05:00	30	TRAFFIC SIGNAL	FUNCTIONING PROPERLY
1	2675c13fd0f474d730a5b780968b3cafc7c12d7adb661f...	None	2023-09-22 18:45:00	50	NO CONTROLS	NO CONTROLS
2	5f54a59fcb087b12ae5b1acff96a3caf4f2d37e79f8db4...	None	2023-07-29 14:45:00	30	TRAFFIC SIGNAL	FUNCTIONING PROPERLY
3	7ebf015016f83d09b321afd671a836d6b148330535d5df...	None	2023-08-09 23:00:00	30	NO CONTROLS	NO CONTROLS
4	6c1659069e9c6285a650e70d6f9b574ed5f64c12888479...	None	2023-08-18 12:50:00	15	OTHER	FUNCTIONING PROPERLY

5 rows × 48 columns

In [4]:
```
# 2) Preparazione dati: estrapolazione, pulizia e ordinamento

# Conversione della colonna con la data dell'incidente
df['CRASH_DATE'] = pd.to_datetime(df['CRASH_DATE'], errors='coerce') # converte la colonna CRASH_DATE in un oggetto data/ora

# toglie le righe con data non valida, come le NaT (Not a Time) generate dell'opzione di sicurezza 'coerce'
df = df.dropna(subset=['CRASH_DATE'])

df = df.sort_values('CRASH_DATE')
df.head()
```

Out[4]:

	CRASH_RECORD_ID	CRASH_DATE_EST_I	CRASH_DATE	POSTED_SPEED_LIMIT	TRAFFIC_CONTROL_DEVICE	DEVICE_CONDI
535086	a802658be15312809c771559e4f81088cfb226830792a5...	None	2013-03-03 16:48:00	30	TRAFFIC SIGNAL	FUNCTIO PRO
115024	19fb5af681f833c2af85734245f737fa6fbe62ac1ea379...	None	2013-06-01 20:29:00	30	NO CONTROLS	NO CONT
765976	f62e27317feb174811cf4fefeb9fa1064fea6c0619a873...	None	2014-01-18 18:14:00	30	NO CONTROLS	NO CONT
480324	957783a4787318f005a7dbc920e4c84cb9ac8aa7329a62...	None	2014-01-21 07:40:00	30	YIELD	NO CONT
124001	1d0232afecbdfd01968555aa956a688fd6f55a2bd1984f...	None	2014-02-24 19:45:00	30	TRAFFIC SIGNAL	FUNCTIO PRO

5 rows × 48 columns

In [5]:
```
# 3) Costruzione della serie storica aggregata

# Granularità: settimanale
ts = (
    df.set_index('CRASH_DATE') # rende CRASH_DATE la colonna indice del DataFrame
      .resample('W') # raggruppa i dati secondo una freq. temporale ('W' = Weekly, con la domenica come termine di default)
      .size() # conta quanti incidenti ci sono in ciascun raggruppamento settimanale
      .rename('num_crashes') # rinomina la colonna del conteggio
      .to_frame() # riconversione in un DataFrame della singola colonna restituita da .size()
)

# riempie eventuali settimane senza incidenti con 0
ts['num_crashes'] = ts['num_crashes'].fillna(0)

ts.head()
```

Out[5]:

CRASH_DATE	num_crashes
2013-03-03	1
2013-03-10	0
2013-03-17	0
2013-03-24	0
2013-03-31	0

In [6]:
```
# 4) Definizionevv Split temporale per il Train/Test

# Selezione della finestra temporale di lavoro
ts = ts.loc['2016-01-01':'2023-12-31'] # .loc seleziona un sottoinsieme della serie storica

# Train: 2016–2021, Test: 2022–2023
split_date = '2022-01-01'

train_ts = ts.loc[:'2021-12-31']
test_ts  = ts.loc[split_date:]
```

In [7]:
```
train_ts.head()
```

Out[7]:

CRASH_DATE	num_crashes
2016-01-03	505
2016-01-10	621
2016-01-17	666
2016-01-24	664
2016-01-31	598

In [8]:
```
test_ts.head()
```

Out[8]:

CRASH_DATE	num_crashes
2022-01-02	1615
2022-01-09	1778
2022-01-16	1806
2022-01-23	1826
2022-01-30	2039

In [9]:
```
# 5) Trasformazione della serie in un problema supervisionato con lag features

# Funzione helper per creare le features di lag e di calendario
def make_supervised(series, n_lags=7, freq='W'):

    # TARGET (output): colonna 'y' dei valori da prevedere
    df_sup = pd.DataFrame({'y': series}) # nuovo dataframe per la colonna target

    # FEATURES (input): colonne 'x' degli input da dare al modello durante l'addestramento
    # Lag features
    for lag in range(1, n_lags + 1):
        df_sup[f'lag_{lag}'] = df_sup['y'].shift(lag)

    # Features di calendario
    idx = df_sup.index
    df_sup['month'] = idx.month
    df_sup['year'] = idx.year

    df_sup = df_sup.dropna()
    return df_sup
```

In [10]:
```
# Train + Test

supervised = make_supervised(ts['num_crashes'], n_lags=7, freq='W') # applica la funzione alla serie storica

# Esegue lo split train/test sul nuovo dataframe supervisionato
train_sup = supervised.loc[supervised.index < split_date]
test_sup  = supervised.loc[supervised.index >= split_date]

# Matrice delle Features (input) per l'addestramento (tutto train_sup tranne la colonna y)
X_train = train_sup.drop(columns=['y'])
# Vettore di Target (output) per l'addestramento (solo la colonna y)
y_train = train_sup['y']

# Stesse operazioni, ma per i set di test
X_test  = test_sup.drop(columns=['y'])
y_test  = test_sup['y']
```

In [11]:
```
X_train.head()
```

Out[11]:

CRASH_DATE	lag_1	lag_2	lag_3	lag_4	lag_5	lag_6	lag_7	month	year
2016-02-21	672.0	546.0	598.0	664.0	666.0	621.0	505.0	2	2016
2016-02-28	620.0	672.0	546.0	598.0	664.0	666.0	621.0	2	2016
2016-03-06	618.0	620.0	672.0	546.0	598.0	664.0	666.0	3	2016
2016-03-13	661.0	618.0	620.0	672.0	546.0	598.0	664.0	3	2016
2016-03-20	659.0	661.0	618.0	620.0	672.0	546.0	598.0	3	2016

In [12]:
```
y_train.head()
```

Out[12]:
```
CRASH_DATE
2016-02-21    620
2016-02-28    618
2016-03-06    661
2016-03-13    659
2016-03-20    666
Freq: W-SUN, Name: y, dtype: int64
```

In [13]:
```
supervised.head()
```

Out[13]:

CRASH_DATE	y	lag_1	lag_2	lag_3	lag_4	lag_5	lag_6	lag_7	month	year
2016-02-21	620	672.0	546.0	598.0	664.0	666.0	621.0	505.0	2	2016
2016-02-28	618	620.0	672.0	546.0	598.0	664.0	666.0	621.0	2	2016
2016-03-06	661	618.0	620.0	672.0	546.0	598.0	664.0	666.0	3	2016
2016-03-13	659	661.0	618.0	620.0	672.0	546.0	598.0	664.0	3	2016
2016-03-20	666	659.0	661.0	618.0	620.0	672.0	546.0	598.0	3	2016

In [14]:
```
X_test.head()
```

Out[14]:

CRASH_DATE	lag_1	lag_2	lag_3	lag_4	lag_5	lag_6	lag_7	month	year
2022-01-02	1667.0	1985.0	2120.0	2090.0	1862.0	2139.0	2095.0	1	2022
2022-01-09	1615.0	1667.0	1985.0	2120.0	2090.0	1862.0	2139.0	1	2022
2022-01-16	1778.0	1615.0	1667.0	1985.0	2120.0	2090.0	1862.0	1	2022
2022-01-23	1806.0	1778.0	1615.0	1667.0	1985.0	2120.0	2090.0	1	2022
2022-01-30	1826.0	1806.0	1778.0	1615.0	1667.0	1985.0	2120.0	1	2022

In [15]:
```
y_test.head()
```

Out[15]:
```
CRASH_DATE
2022-01-02    1615
2022-01-09    1778
2022-01-16    1806
2022-01-23    1826
2022-01-30    2039
Freq: W-SUN, Name: y, dtype: int64
```

In [16]:
```
# 6) Creazione modello, Addestramento e Previsione

rf = RandomForestRegressor(
    n_estimators=300, # numero di alberi decisionali
    random_state=42, # seme casuale per la riproducibilità imparziale dell'analisi
    n_jobs=-1 # permette l'uso di tutti i core della CPU per addestrare i 300 alberi in parallelo
)

rf.fit(X_train, y_train) # Training
y_pred = rf.predict(X_test) # Testing e calcolo delle previsioni
```

In [17]:
```
df_pred = pd.DataFrame({
    'y_actual': y_test.values,
    'y_pred': y_pred
}, index=y_test.index)

df_pred.head()
```

Out[17]:

CRASH_DATE	y_actual	y_pred
2022-01-02	1615	1793.746667
2022-01-09	1778	1721.363333
2022-01-16	1806	1886.663333
2022-01-23	1826	1943.440000
2022-01-30	2039	1922.643333

In [18]:
```
# 7) Valutazione del modello (metriche MAE / RMSE)
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
```

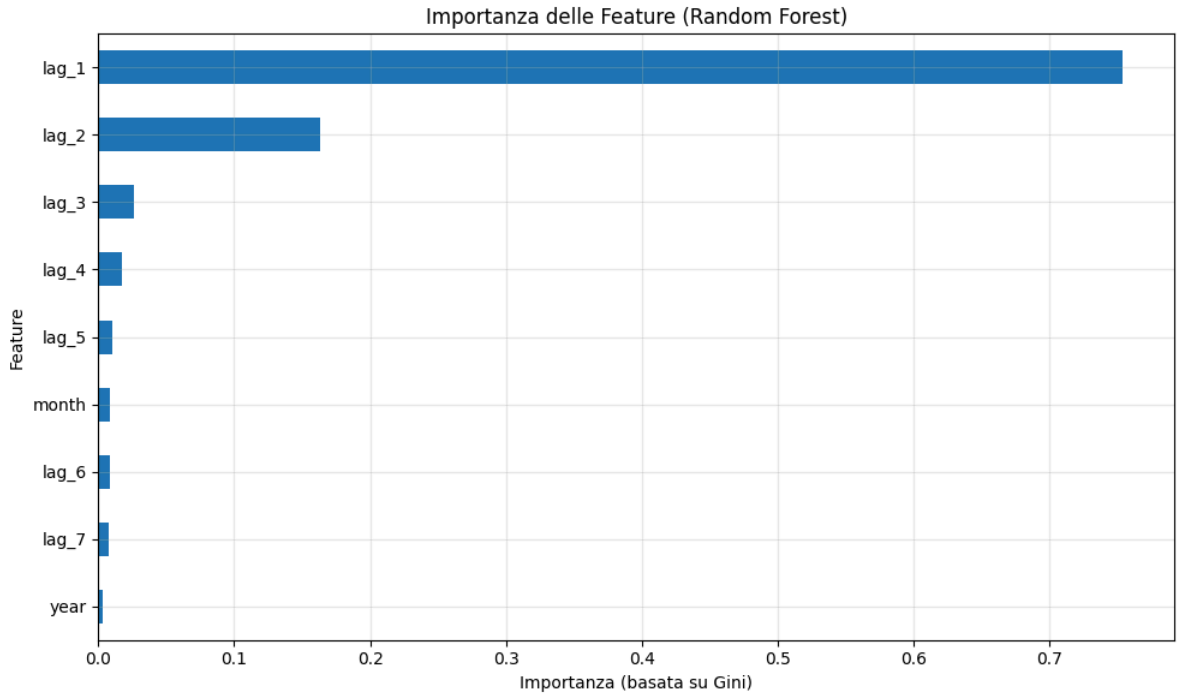
```
print("MAE:", mae)
print("RMSE:", rmse)
```

```
MAE: 125.56631746031746
RMSE: 170.16897522585597
```

In [19]:
```
# 7.1) Calcolo Baseline "Naive"

# Crea la previsione "naive": valore di questa settimana = valore della settimana precedente
y_pred_naive = y_test.shift(1)

# rimuove il primo valore  da entrambe le serie per allinearle
y_test_aligned = y_test.iloc[1:] # toglie il primo valore per avere una serie della stessa lunghezza di y_pred_naive_aligned
y_pred_naive_aligned = y_pred_naive.iloc[1:] # toglie il primo valore poichè è un NaN prodotto dallo shift(1)

# Calcola il MAE del modello "naive"
mae_naive = mean_absolute_error(y_test_aligned, y_pred_naive_aligned)

print(f"MAE Modello (Random Forest): {mae:.2f}")
print(f"MAE Baseline (Naive): {mae_naive:.2f}")

# Confronto
if mae < mae_naive:
    print("\nRISULTATO: Il modello Random Forest è migliore del baseline naive.")
    print(f"È più accurato di circa {mae_naive - mae:.2f} incidenti a settimana.")
else:
    print("\nATTENZIONE: Il modello Random Forest è peggiore (o uguale) del baseline naive.")
```

```
MAE Modello (Random Forest): 125.57
MAE Baseline (Naive): 152.87

RISULTATO: Il modello Random Forest è migliore del baseline naive.
È più accurato di circa 27.30 incidenti a settimana.
```

In [20]:
```
# 7.2) Analisi Importanza Feature

# Estrae i contributi delle Features dal modello
importances = rf.feature_importances_
feature_names = X_train.columns # nomi delle features

# Crea una Series pandas per visualizzarle facilmente
feat_imp = pd.Series(importances, index=feature_names).sort_values()

# Crea il grafico
plt.figure(figsize=(10, 6))
feat_imp.plot(kind='barh', title='Importanza delle Feature (Random Forest)')
plt.xlabel('Importanza (basata su Gini)')
plt.ylabel('Feature')
plt.grid(alpha=0.3)
plt.tight_layout() # per non tagliare le etichette
plt.show()
```

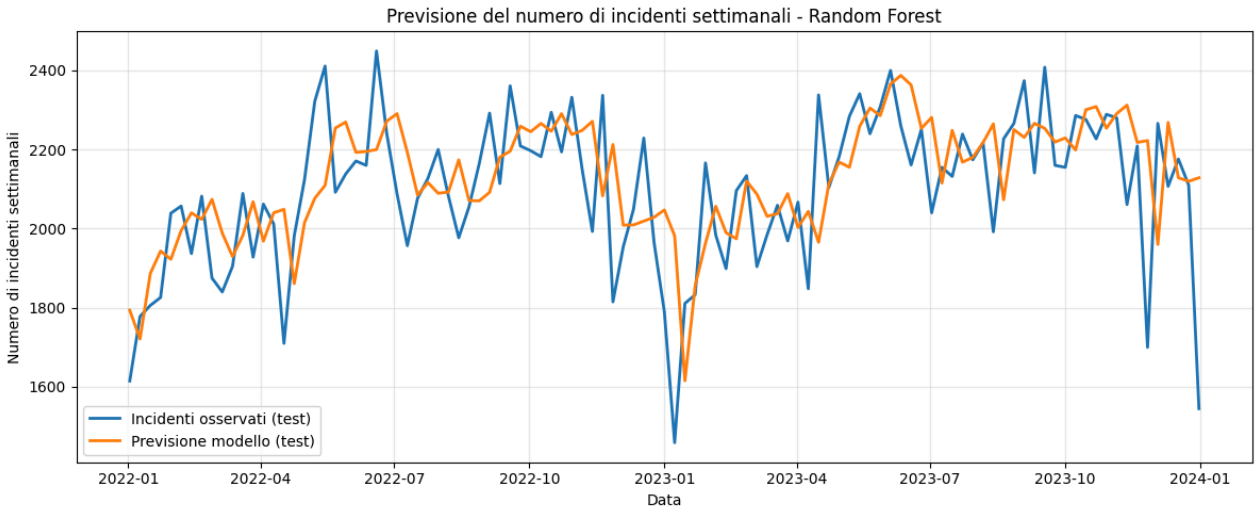


In [21]:
```
# Per comodità crea un nuovo dataframe che allinea per data i valori reali (y_actual) e i valori previsti (y_pred)
results = pd.DataFrame({
    'date': test_sup.index,
    'y_actual': y_test,
    'y_pred': y_pred
}).set_index('date')
results.head()
```

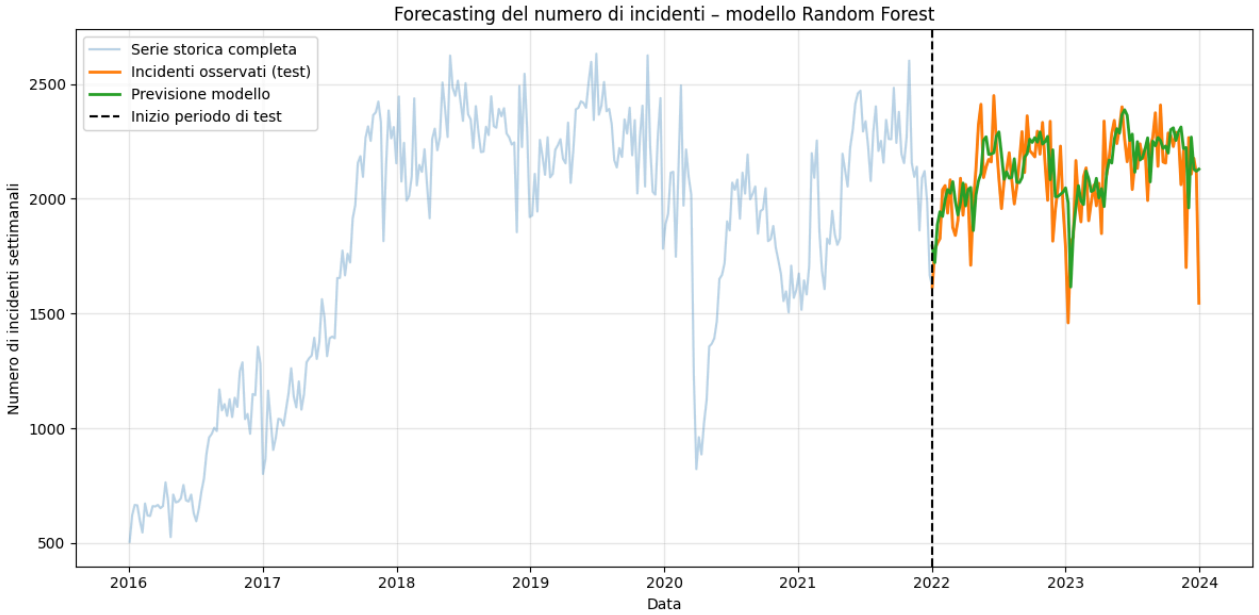
Out[21]:

date	y_actual	y_pred
2022-01-02	1615	1793.746667
2022-01-09	1778	1721.363333
2022-01-16	1806	1886.663333
2022-01-23	1826	1943.440000
2022-01-30	2039	1922.643333

In [22]:
```
# 8) Grafici con matplotlib

# 8.1) Grafico col solo intervallo di Test
plt.figure(figsize=(12, 5))

plt.plot(results.index, results['y_actual'],
         label='Incidenti osservati (test)', linewidth=2)
plt.plot(results.index, results['y_pred'],
         label='Previsione modello (test)', linewidth=2)

plt.xlabel('Data')
plt.ylabel('Numero di incidenti settimanali')
plt.title('Previsione del numero di incidenti settimanali - Random Forest')
plt.legend()
plt.grid(alpha=0.3)
plt.tight_layout()
plt.show()
```



In [23]:
```
# 8.2) Grafico con il contesto completo Train + Test

plt.figure(figsize=(12, 6))

plt.plot(ts.index, ts['num_crashes'],
         label='Serie storica completa', alpha=0.3)
plt.plot(results.index, results['y_actual'],
         label='Incidenti osservati (test)', linewidth=2)
plt.plot(results.index, results['y_pred'],
         label='Previsione modello', linewidth=2)

split_dt = pd.to_datetime(split_date)
plt.axvline(split_dt, linestyle='--', color='k', label='Inizio periodo di test')

plt.xlabel('Data')
plt.ylabel('Numero di incidenti settimanali')
plt.title('Forecasting del numero di incidenti – modello Random Forest')
plt.legend()
plt.grid(alpha=0.3)
plt.tight_layout()
plt.show()
```



In [24]:
```python
# 9) Allenamento del modello finale su tutti i dati (2016 - 2023)

# Usa l'intero dataframe 'supervised' creato in precedenza
X_final = supervised.drop(columns=['y'])
y_final = supervised['y']

# Crea e allena il modello finale
rf_final = RandomForestRegressor(
    n_estimators=300,
    random_state=42,
    n_jobs=-1
)

rf_final.fit(X_final, y_final)
```

Out[24]:

RandomForestRegressor

▶ Parameters

In [25]:
```python
# 10) Generazione della previsione autoregressiva

N_LAGS = 7          # stesso n_lags usato prima
N_FORECAST = 52     # numero di settimane da prevedere (circa 1 anno)

# Prende gli ultimi lag reali per iniziare la previsione
history_lags = y_final.iloc[-N_LAGS:].tolist()

# Crea l'indice di date per il futuro
last_real_date = y_final.index[-1]
future_dates = pd.date_range(
    start=last_real_date + pd.Timedelta(weeks=1),
    periods=N_FORECAST,
    freq='W'
)

# Lista per salvare le previsioni
future_forecasts = []

print(f"Inizio previsione autoregressiva per {N_FORECAST} settimane...")

for current_date in future_dates:
    # 1- Costruisce le features per la data corrente
    features = {}
    for i in range(1, N_LAGS + 1):
        features[f'lag_{i}'] = history_lags[-i]
    features['month'] = current_date.month
    features['year'] = current_date.year

    # 2- Crea un dataframe per la previsione
    X_new = pd.DataFrame(features, index=[current_date])
    X_new = X_new[X_final.columns]

    # 3- Previsione
    new_pred = rf_final.predict(X_new)[0] # [0] per estrarre il singolo valore

    # 4- Salva la previsione e aggiorna la history
    future_forecasts.append(new_pred)
    history_lags.append(new_pred) # La previsione diventa 'storia'
    history_lags = history_lags[-N_LAGS:] # Mantiene solo gli ultimi 7 valori

print("Previsione completata.")
```

```
Inizio previsione autoregressiva per 52 settimane...
Previsione completata.
```

In [26]:
```python
# 11) Grafico della previsione futura (numero di incidenti nel 2024)
```

```python
# Trasforma le previsioni in una Series Pandas per costruire più facilmente il grafico
s_forecast = pd.Series(future_forecasts, index=future_dates)

plt.figure(figsize=(14, 7))

plt.plot(y_final, label='Dati storici osservati', color='C0', alpha=0.8) # Dati storici
plt.plot(s_forecast, label='Previsione futura (2024-2025)', color='C3', linestyle='-') # Previsione futura
plt.axvline(last_real_date, linestyle=':', color='k', label='Inizio previsione')

plt.title('Previsione Autoregressiva Incidenti Settimanali')
plt.ylabel('Numero di incidenti settimanali')
plt.xlabel('Data')
plt.legend()
plt.grid(alpha=0.3)
plt.tight_layout()
plt.show()
```