In [28]:
```
# Python imports
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import os
import numpy as np

# Pipelines
from sklearn.pipeline import Pipeline

# Preprocessing
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

# 4 models
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier

# Splitting, scoring, tuning
from sklearn.dummy import DummyClassifier
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score, f1_score,average_precision_score,log_los
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

Datová sada - California Weather and Fire Prediction Dataset (1984-2025)

Pro účely analýzy jsem použil dataset s názvem „CA_Weather_Fire_Dataset_1984-2025.csv“. Jedná se o datovou sadu kombinující data o počasí (NOAA) a záznamy o požárech (CAL FIRE). Data jsou veřejně dostupná: odkaz na dataset. V souboru jsou denní záznamy z Kalifornie od 1. ledna 1984 až do 12. ledna 2025. Celkem 14 990 záznamů.

Seznam sloupců v datové sadě:

- **DATE: Datum pozorování (YYYY-MM-DD)
- *PRECIPITATION**: Denní úhrn srážek v palcích
- **MAX_TEMP**: Maximální denní teplota ve °F
- **MIN_TEMP**: Minimální denní teplota ve °F
- **AVG_WIND_SPEED**: Průměrná denní rychlost větru v mph
- **FIRE_START_DAY**: Binární příznak (True/False), zda v daný den začal požár
- **YEAR**: Rok
- **TEMP_RANGE**: MAX_TEMP - MIN_TEMP
- **WIND_TEMP_RATIO**: AVG_WIND_SPEED / MAX_TEMP
- **MONTH**: Kalendářní měsíc (1–12)

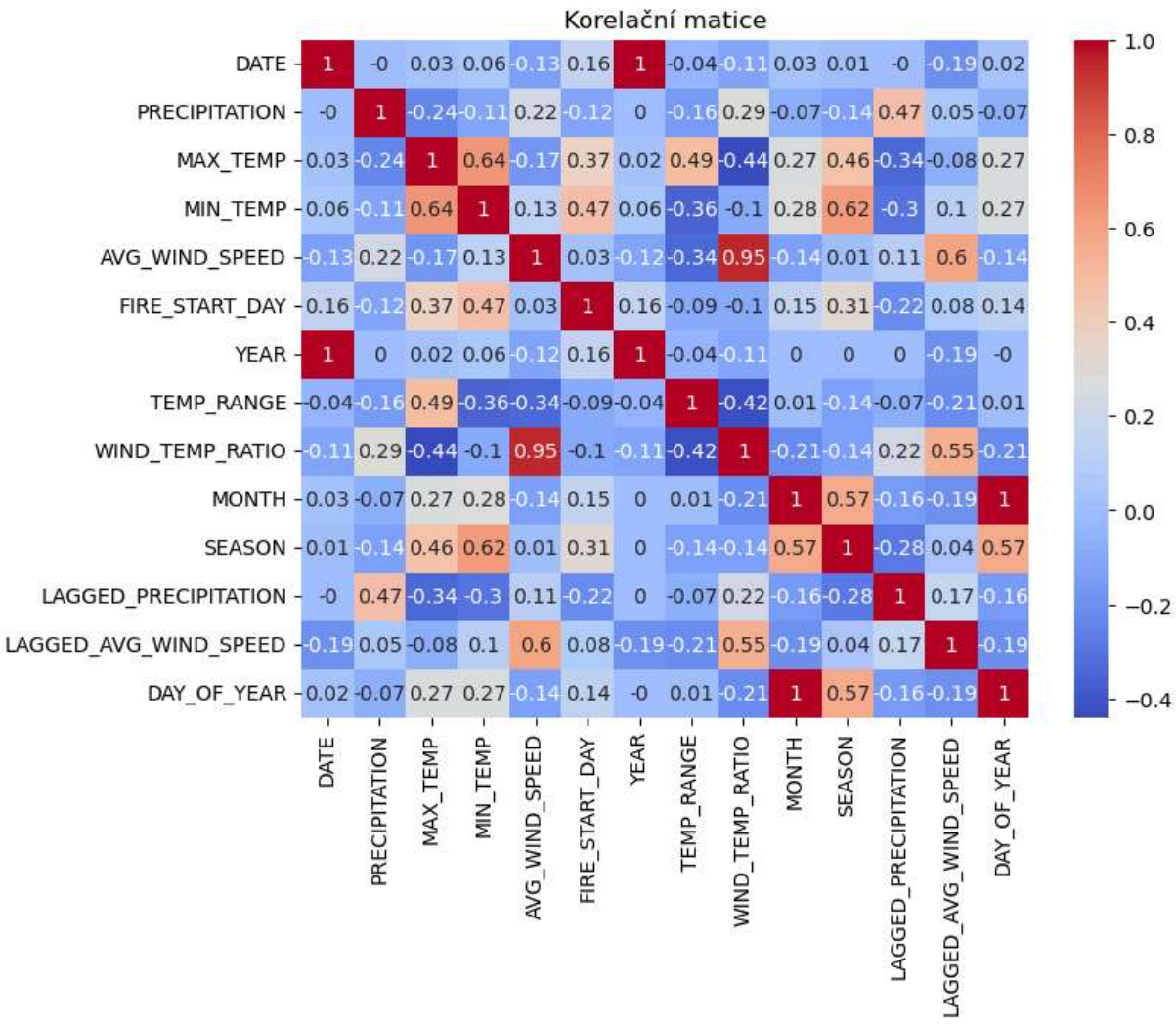
- **SEASON**: Roční období (Winter, Spring, Summer, Fall)
- **LAGGED_PRECIPITATION**: Odvozeno (7denní kumulovaná srážková hodnota)
- **LAGGED_AVG_WIND_SPEED**: Odvozeno (7denní průměrná rychlost větru)
- **DAY_OF_YEAR**: Pořadové číslo dne v roce (1–366)

Načtení dat z csv a základní analýza

In [29]:
```
df = pd.read_csv('CA_Weather_Fire_Dataset_1984-2025.csv')  # Načtení dat do datafra
df['DATE'] = pd.to_datetime(df['DATE'])  # Převedení na datetime
df['SEASON'] = df['DATE'].dt.month % 12 // 3 + 1  # Převedení na numerické hodnoty
df = df[df['YEAR'] < 2024]
```

Korelační heatmapa

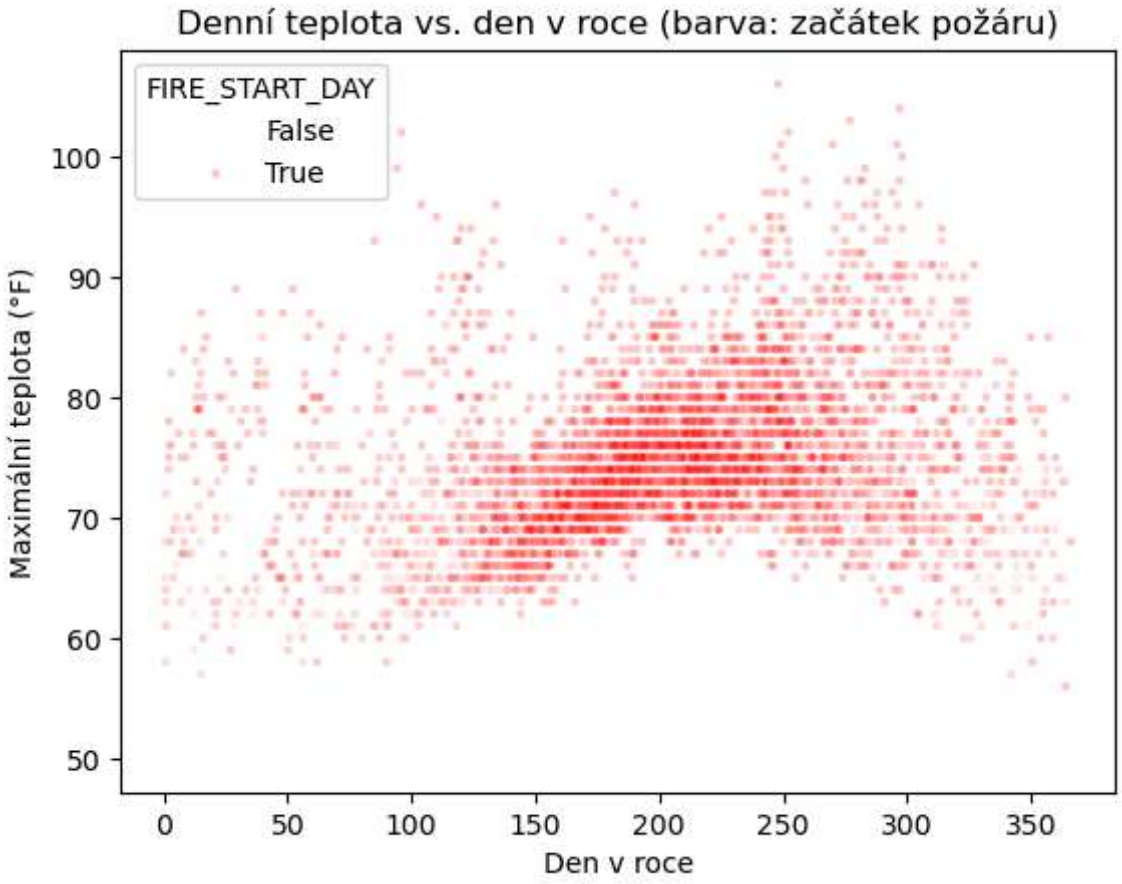
In [30]:
```
def plot_correlation_heatmap(df):
    # Korelační heatmapa
    plt.figure(figsize=(8, 6))
    corr = df.corr().round(2)
    sns.heatmap(corr, annot=True, cmap='coolwarm')
    plt.title('Korelační matice')
    plt.show()

plot_correlation_heatmap(df)
```



Z heatmapy je vidět, že přítomnost požárů vyjádřená proměnné FIRE_START_DAY koreluje s teplotou a ročním obdobím. Negativní korelace je pak s přítomností srážek v proměnné LAGGED PRECIPITATION.

Vztah vzniku požárů na maximální teplotě a dni v roce

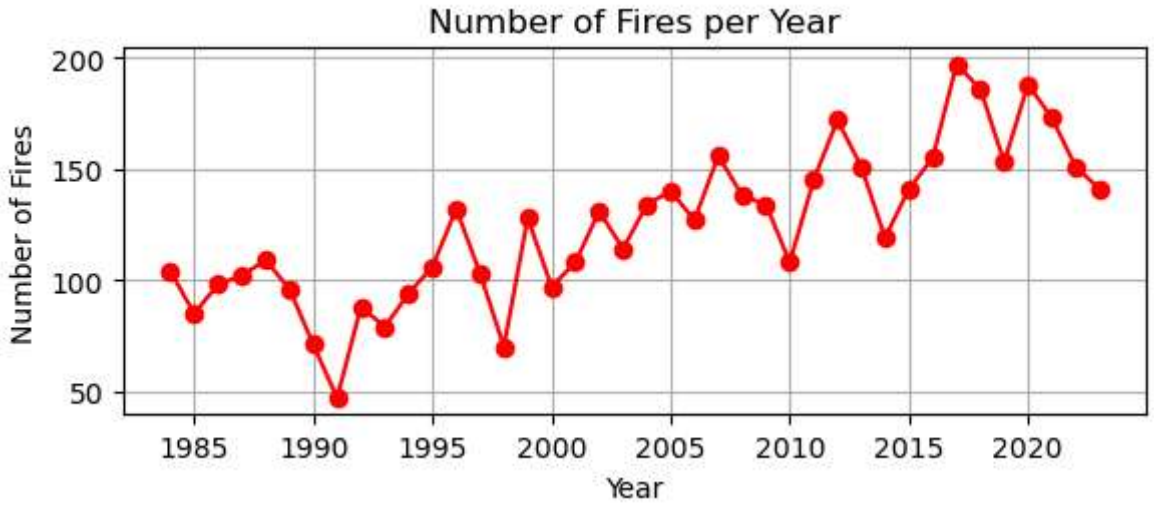
In [31]:
```
sns.scatterplot(data=df, x='DAY_OF_YEAR', y='MAX_TEMP', hue='FIRE_START_DAY', palet
plt.title('Denní teplota vs. den v roce (barva: začátek požáru)')
plt.xlabel('Den v roce')
plt.ylabel('Maximální teplota (°F)')
plt.show()
```

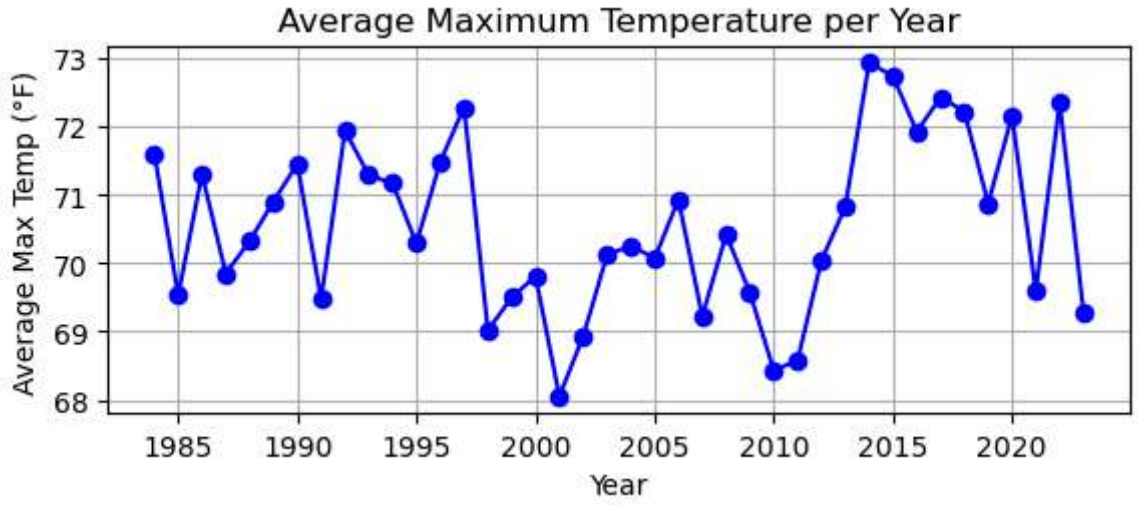


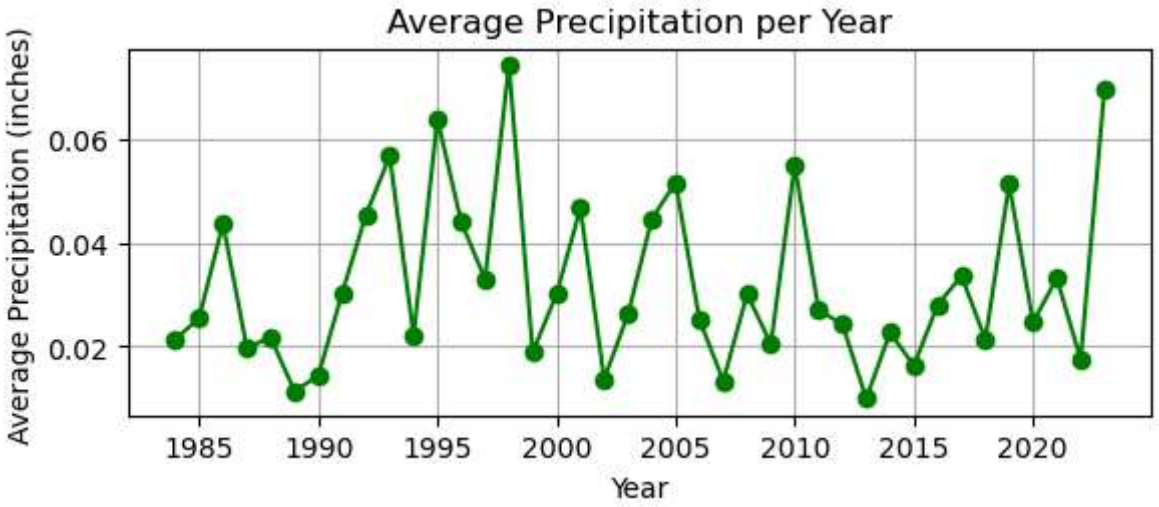
Časové řady počtu výskytů požárů, teploty a srážek

In [32]:
```
# Group data by year for fires and temperature
fires_per_year = df.groupby('YEAR')['FIRE_START_DAY'].sum()
avg_temp_per_year = df.groupby('YEAR')['MAX_TEMP'].mean()
avg_precip_per_year = df.groupby('YEAR')['PRECIPITATION'].mean()

# Create subplots
fig, (ax1, ax2, ax3) = plt.subplots(3, 1, figsize=(6, 8))

# Plot number of fires per year
ax1.plot(fires_per_year.index, fires_per_year.values, marker='o', color='red')
ax1.set_title('Number of Fires per Year')
ax1.set_xlabel('Year')
ax1.set_ylabel('Number of Fires')
ax1.grid(True)

# Plot average temperature per year
ax2.plot(avg_temp_per_year.index, avg_temp_per_year.values, marker='o', color='blue
ax2.set_title('Average Maximum Temperature per Year')
ax2.set_xlabel('Year')
ax2.set_ylabel('Average Max Temp (°F)')
ax2.grid(True)

# Plot average precipitation per year
ax3.plot(avg_precip_per_year.index, avg_precip_per_year.values, marker='o', color='
ax3.set_title('Average Precipitation per Year')
```

```
ax3.set_xlabel('Year')
ax3.set_ylabel('Average Precipitation (inches)')
ax3.grid(True)

plt.tight_layout()
plt.show()
```







Prepare dataset

In [33]:
```
# Create cyclic features for day of year
day = df['DATE'].dt.dayofyear
df['day_sin'] = np.sin(2 * np.pi * day / 365)
df['day_cos'] = np.cos(2 * np.pi * day / 365)
df_model = df.dropna().sort_values('DATE').reset_index(drop=True)
```

Out[33]:

	DATE	PRECIPITATION	MAX_TEMP	MIN_TEMP	AVG_WIND_SPEED	FIRE_START_DAY
0	1984-01-01	0.00	79.0	51.0	4.70	False
1	1984-01-02	0.00	71.0	46.0	5.59	False
2	1984-01-03	0.00	70.0	47.0	5.37	False
3	1984-01-04	0.00	76.0	45.0	4.70	False
4	1984-01-05	0.00	74.0	49.0	5.14	False
...	...	...	...	...	...	...
14602	2023-12-27	0.00	62.0	52.0	5.59	False
14603	2023-12-28	0.00	64.0	50.0	5.59	False
14604	2023-12-29	0.00	65.0	52.0	3.80	False
14605	2023-12-30	0.62	62.0	54.0	8.50	False
14606	2023-12-31	0.00	60.0	51.0	4.92	False

14607 rows × 16 columns

VIF

In [8]:
```
def calculate_vif(dataframe, exclude_columns=None):
    """
    Calculate Variance Inflation Factor (VIF) for numeric columns in a DataFrame.
    """
    numeric_cols = dataframe.select_dtypes(include=[np.number]).columns.tolist()
    if exclude_columns:
        numeric_cols = [col for col in numeric_cols if col not in exclude_columns]
    df_vif = dataframe[numeric_cols].dropna()
```

```
    vif_data = pd.DataFrame({
        "Variable": df_vif.columns,
        "VIF": [variance_inflation_factor(df_vif.values, i) for i in range(df_vif.s
    })
    return vif_data.sort_values('VIF', ascending=False).reset_index(drop=True)


# Initial VIF calculation
exclude_columns_full = ['FIRE_START_DAY', 'DATE']
vif_result = calculate_vif(df_model, exclude_columns=exclude_columns_full)
print("VIF Analysis: before removing high VIF columns")
print(vif_result)
print("---")
print("VIF Analysis: after removing high VIF columns")
exclude_columns = ['FIRE_START_DAY', 'DATE', 'MIN_TEMP', 'DAY_OF_YEAR', 'WIND_TEMP_
vif_result = calculate_vif(df, exclude_columns=exclude_columns)
print(vif_result)
```

```
c:\Users\esiff\anaconda3\Lib\site-packages\statsmodels\stats\outliers_influence.py:1
97: RuntimeWarning: divide by zero encountered in scalar divide
  vif = 1. / (1. - r_squared_i)
VIF Analysis: before removing high VIF columns
                 Variable          VIF
0                MAX_TEMP          inf
1                MIN_TEMP          inf
2              TEMP_RANGE          inf
3          AVG_WIND_SPEED  1113.909246
4                    YEAR  1013.315532
5         WIND_TEMP_RATIO   978.028888
6                   MONTH   661.459700
7             DAY_OF_YEAR   578.237709
8   LAGGED_AVG_WIND_SPEED    71.112195
9                  SEASON    17.828567
10               day_sin     4.713895
11               day_cos     4.356023
12   LAGGED_PRECIPITATION     1.881356
13          PRECIPITATION     1.490688
---
VIF Analysis: after removing high VIF columns
               Variable        VIF
0        AVG_WIND_SPEED  13.371746
1              MAX_TEMP  12.633336
2  LAGGED_PRECIPITATION   1.603841
3         PRECIPITATION   1.395610
4               day_cos   1.187150
5               day_sin   1.117751
```

Pomocné funkce

In [ ]:
```
# Split the data to train and test sets with only selected features
def train_test_split_df(df):
    y = df['FIRE_START_DAY']
    X = df.drop(columns=exclude_columns)
    return train_test_split(X, y, test_size=0.2, shuffle=True, random_state=42, str
```

```python
# evaluate model performance
def evaluate_model(name,model, X_train, X_test, y_train, y_test, threshold=0.5):
    print(f"Evaluating {name}...")

    try:
        y_proba_train = model.predict_proba(X_train)[:, 1]
        y_pred_train  = (y_proba_train > threshold).astype(int)
        y_proba_test  = model.predict_proba(X_test)[:, 1]
        y_pred_test   = (y_proba_test >= threshold).astype(int)

        metrics_train = {
            'model': name + ' train',
            'threshold': threshold,
            'roc_auc': roc_auc_score(y_train, y_proba_train),
            'pr_auc': average_precision_score(y_train, y_proba_train),
            'f1': f1_score(y_train, y_pred_train),
            'accuracy': accuracy_score(y_train, y_pred_train),
            'log_loss': log_loss(y_train, y_proba_train)
        }
        metrics_test = {
            'model': name+ ' test',
            'threshold': threshold,
            'roc_auc': roc_auc_score(y_test, y_proba_test),
            'accuracy': accuracy_score(y_test, y_pred_test),
            'pr_auc': average_precision_score(y_test, y_proba_test),
            'f1': f1_score(y_test, y_pred_test),
            'log_loss': log_loss(y_test, y_proba_test)
        }
        return metrics_train, metrics_test
    except Exception as e:
        print(f"Error evaluating {model.name}: {e}")
        return None, None
```

Baseline Pipelines

In [19]:

```python
# Pipeline for Logistic Regression
pipe_logreg = Pipeline([
    ('scaler', StandardScaler()),
    ('clf', LogisticRegression())
])

pipe_svc = Pipeline([
    ('scaler', StandardScaler()),
    ('clf', SVC(probability=True)) # Enable probability estimates, default is only
])

pipe_rf = Pipeline([
    ('scaler', StandardScaler()),
    ('clf', RandomForestClassifier())
])

pipe_gb = Pipeline([
    ('scaler', StandardScaler()),
    ('clf', GradientBoostingClassifier())
])
```

```
pipe_false = Pipeline([
    ('scaler', StandardScaler()),
    ('clf', DummyClassifier(strategy='most_frequent'))
])

ms = {
    'allways_false': pipe_false,
    'logreg': pipe_logreg,
    'svc': pipe_svc,
    'rf': pipe_rf,
    'gb': pipe_gb,
}
```

In [87]:
```
# evaluate all baseline models
X_train, X_test, y_train, y_test = train_test_split_df(df_model)
results = []
for name, pipe in ms.items():
    pipe.fit(X_train, y_train)
    tm, te = evaluate_model(name,pipe, X_train, X_test, y_train, y_test)
    if tm and te:
        results.extend([tm, te])

results_df = pd.DataFrame(results).sort_values('f1', ascending=False)
results_df.to_csv('model_results_baseline.csv', index=False)
print(results_df.head(10).sort_values('f1', ascending=False))
```

```
Evaluating allways_false...
Evaluating logreg...
Evaluating svc...
Evaluating rf...
Evaluating gb...
                 model  threshold   roc_auc    pr_auc        f1  accuracy  \
6             rf train        0.5  0.999712  0.999435  0.985877  0.990329
8             gb train        0.5  0.857552  0.739290  0.692259  0.792811
2         logreg train        0.5  0.839229  0.699548  0.679193  0.780830
4            svc train        0.5  0.838553  0.701012  0.678226  0.780830
5             svc test        0.5  0.825004  0.674547  0.656883  0.767967
3          logreg test        0.5  0.826237  0.674400  0.648593  0.764887
9              gb test        0.5  0.827753  0.673133  0.647902  0.764545
7              rf test        0.5  0.796963  0.625412  0.600106  0.740931
1   allways_false test        0.5  0.500000  0.340178  0.000000  0.659822
0  allways_false train        0.5  0.500000  0.340351  0.000000  0.659649

    log_loss
6   0.132804
8   0.440846
2   0.464291
4   0.465513
5   0.481451
3   0.479519
9   0.479045
7   0.599784
1  12.261256
0  12.267489
```

Ladění modelů s parametry a přidanými daty z předchozích dní

In [55]:
```python
# Define parameter grids for each model

from sklearn.model_selection import GridSearchCV
param_grids = {
    'logreg': {
        'clf__C': [0.1, 1,10],              # default C=1
        'clf__penalty': ['l1', 'l2'],        # default penalty='l2'
        'clf__solver': ['liblinear', 'saga'] # default solver='lbfgs' (note: 'libli
    },
    'svc': {
        'clf__C': [1,3,5],                   # default C=1
        'clf__gamma': ['scale', 0.01, 0.1, 1], # default gamma='scale'
        'clf__kernel': ['rbf', 'linear']     # default kernel='rbf'
    },
    'rf': {
        'clf__n_estimators': [50, 100, 200], # default n_estimators=100
        'clf__max_depth': [None, 10, 30],# default max_depth=None
        'clf__min_samples_split': [2, 5, 10],# default min_samples_split=2
        'clf__min_samples_leaf': [1, 2, 4]   # default min_samples_leaf=1
    },
    'gb': {
        'clf__n_estimators': [50, 100, 200], # default n_estimators=100
        'clf__learning_rate': [0.01, 0.1, 0.2], # default learning_rate=0.1
        'clf__max_depth': [3, 5, 7],          # default max_depth=3
        'clf__min_samples_split': [2, 5, 10]  # default min_samples_split=2
    }
}
```

In [80]:
```python
# Extend dataset with lag features by specified number of days
def create_extended_features(df_model, lag_days=3):
    df_model_extended = df_model.copy()
    cols_to_lag = [
        'MAX_TEMP',
        'PRECIPITATION',
        'AVG_WIND_SPEED',
        'LAGGED_PRECIPITATION',
        'day_sin',
        'day_cos',
    ]
    lags = range(1, lag_days + 1)
    for lag in lags:
        for col in cols_to_lag:
            df_model_extended[f"{col}_{lag}_DAY_AGO"] = df_model_extended[col].shif
    # Remove rows with NaN values (caused by shifting)
    df_model_extended.dropna(inplace=True)
    df_model_extended.reset_index(drop=True, inplace=True)
    print(f"Created extended dataset with {len(df_model_extended)} rows for {lag_da
    return df_model_extended
```

In [ ]:
```python
# Grid search with cross-validation for hyperparameter tuning for different lag day
# runs a few hours
```

```
# Reuse for the testing of the best model
results_cv = []
max_lag_days = 5 # Test lag features from 0 to max_lag_days

for i in range(max_lag_days+1):  # From 0 to max_lag_days
    df = create_extended_features(df_model, lag_days=i)
    X_train, X_test, y_train, y_test = train_test_split_df(df) #custom function tes
    print(f"--- Using {i} days of lag features ---")
    for name, pipe in ms.items():
        if name == 'allways_false':  # Skip dummy classifier
            continue
        if name in param_grids:
            grid_search = GridSearchCV(
                pipe,
                param_grids[name],
                cv=5,
                scoring='f1',
                n_jobs=-1,  # Use all available cores
                verbose=0 # dont print progress
            )
            grid_search.fit(X_train, y_train)
        # Store CV results
        cv_result = {
            'lag_days': i,
            'model': name,
            'best_cv_f1': grid_search.best_score_,
            'best_params': grid_search.best_params_
        }
        results_cv.append(cv_result)

cv_results_df = pd.DataFrame(results_cv)
cv_results_df = cv_results_df.sort_values(by=['best_cv_f1'], ascending=False).reset
cv_results_df.to_csv(f'cv_results_a{max_lag_days}.csv', index=False)
```

In [89]:

```
# Show top 10 models, parameters and number of lag days with default threshold
results_df = pd.read_csv('cv_results_5.csv')
print(results_df.sort_values('best_cv_f1',ascending=False).head(10))
```

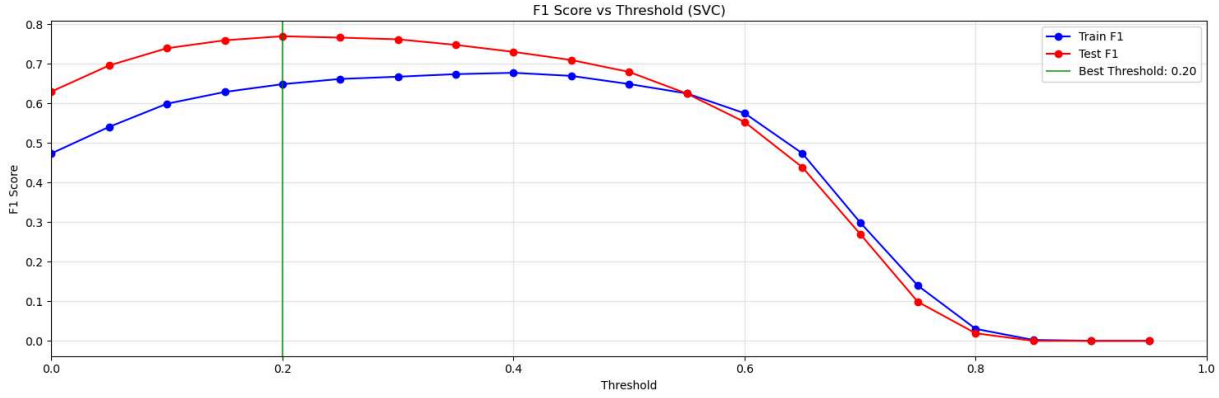
```
   lag_days   model  best_cv_f1  \
0         5     svc    0.695878
1         4     svc    0.693543
2         0     svc    0.692076
3         3     svc    0.691962
4         2     svc    0.690097
5         1     svc    0.687276
6         3  logreg    0.686741
7         5  logreg    0.684935
8         4  logreg    0.684475
9         0  logreg    0.680565

                                         best_params
0  {'clf__C': 10, 'clf__gamma': 'scale', 'clf__ke...
1  {'clf__C': 10, 'clf__gamma': 'scale', 'clf__ke...
2  {'clf__C': 1, 'clf__gamma': 'scale', 'clf__ker...
3  {'clf__C': 1, 'clf__gamma': 'scale', 'clf__ker...
4  {'clf__C': 1, 'clf__gamma': 'scale', 'clf__ker...
5  {'clf__C': 1, 'clf__gamma': 'scale', 'clf__ker...
6  {'clf__C': 0.1, 'clf__penalty': 'l2', 'clf__so...
7  {'clf__C': 0.1, 'clf__penalty': 'l1', 'clf__so...
8  {'clf__C': 0.1, 'clf__penalty': 'l2', 'clf__so...
9  {'clf__C': 0.1, 'clf__penalty': 'l1', 'clf__so...
```

Nalezení prahu nejlepšího modelu

In [ ]:
```python
# ------------------- Best parameters from the grid search ------------------- #
lag_days = 5
model = 'svc'
params = {'clf__C': 3, 'clf__gamma': 'scale', 'clf__kernel': 'linear'}

# ------------------ Test threshold vs F1 for the best model ----------------- #
df_extended = create_extended_features(df_model, lag_days)
X_train, X_test, y_train, y_test = train_test_split_df(df_extended)
best_pipe = ms[model]
best_pipe.set_params(**params)
results_best=[]
best_pipe.fit(X_train, y_train)
for threshold in np.arange(0.0, 1, 0.05):
    print(f"--- Evaluating with threshold {threshold:.1f} ---")
    tm, te = evaluate_model(model, best_pipe, X_train, X_test, y_train, y_test, thr
    if tm and te:
        results_best.extend([tm, te])
results_df_best = pd.DataFrame(results_best).sort_values('roc_auc', ascending=False
results_df_best.to_csv(f'model_results_{model}_{lag_days}.csv', index=False)
print(results_df_best)
```

In [ ]:
```python
# Plot Threshold vs F1 Score
results_df_best.sort_values('threshold', ascending=False, inplace=True)
fig, (ax1) = plt.subplots(figsize=(15, 5))

train_results = results_df_best[results_df_best['model'].str.contains('train')]
test_results = results_df_best[results_df_best['model'].str.contains('test')]

# Plot 1: F1 Score vs Threshold
```

```
ax1.plot(train_results['threshold'], train_results['f1'], 'o-', label='Train F1', c
ax1.plot(test_results['threshold'], test_results['f1'], 'o-', label='Test F1', colo
ax1.set_xlabel('Threshold')
ax1.set_ylabel('F1 Score')
ax1.set_title(f'F1 Score vs Threshold ({model.upper()})')
ax1.legend()
ax1.grid(True, alpha=0.3)
ax1.set_xlim(0, 1)

# Find best threshold for test F1
best_test_idx = test_results['f1'].idxmax()
best_threshold = test_results.loc[best_test_idx, 'threshold']
best_f1 = test_results.loc[best_test_idx, 'f1']
ax1.axvline(x=best_threshold, color='green',  alpha=0.7, label=f'Best Threshold: {b
ax1.legend()

plt.tight_layout()
plt.show()
```



In [ ]:
```
# Threshold vs F1 for the best model
results_df = pd.read_csv('model_results_svc_5.csv')
print(results_df.sort_values('f1',ascending=False).head(5))
```

	model	threshold	roc_auc	pr_auc	f1	accuracy	log_loss
7	svc test	0.20	0.851399	0.812786	0.770161	0.765834	0.546126
6	svc test	0.25	0.851399	0.812786	0.766667	0.774735	0.546126
4	svc test	0.30	0.851399	0.812786	0.762435	0.780897	0.546126
2	svc test	0.15	0.851399	0.812786	0.759898	0.742554	0.546126
5	svc test	0.35	0.851399	0.812786	0.748252	0.778158	0.546126

Vyhodnocení

Baseline modely

- Při testování porovnáním f1 score baseline modelů vyšla nejlépe logistická regrese se score 0.648593
- Accuracy měl nejlepší model svm 76,6% , což převyšovalo jednoduchý dummy model hádající pokaždé False, který měl 65,9%.

In [81]:
```
results_df = pd.read_csv('model_results_baseline.csv')
print(results_df.sort_values('f1',ascending=False))
```

```
                model   roc_auc    pr_auc        f1  accuracy   log_loss
0            rf train  0.999669  0.999356  0.985852  0.990329   0.133207
1            gb train  0.857552  0.739290  0.692259  0.792811   0.440846
2        logreg train  0.839229  0.699548  0.679193  0.780830   0.464291
5           svc train  0.818316  0.689971  0.661010  0.781172   0.484502
4         logreg test  0.826237  0.674400  0.648593  0.764887   0.479519
3            gb test  0.827753  0.673133  0.647902  0.764545   0.479045
7           svc test  0.797260  0.656581  0.631806  0.766256   0.507625
6            rf test  0.799532  0.635189  0.603913  0.743669   0.593647
8  allways_false test  0.500000  0.340178  0.000000  0.659822  12.261256
9 allways_false train  0.500000  0.340351  0.000000  0.659649  12.267489
```

Parametry nejlepšího modelu

Na základě grid search analýzy byl identifikován nejlepší model s následujícími charakteristikami:

- Model: Support Vector Machine (SVM)
- Lag days: 5 dní historických dat
- C = 3
- gamma = 'scale'
- kernel = 'linear'
- probability = True

Výkon modelu:

- Test F1 Score: 0,770 (při optimálním prahu)/ 0,696 (při defaultním prahu)
- Optimální threshold: 0.2
- Nejvyšší Accuracy měl model 78,1% při prahu 0,3.