sales.py

```python
# Importing necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split

# Load data
train = pd.read_csv("../input/tabular-playground-series-jan-2022/train.csv",
index_col="row_id")
test = pd.read_csv("../input/tabular-playground-series-jan-2022/test.csv",
index_col="row_id")
sample = pd.read_csv("../input/tabular-playground-series-jan-2022/sample_submission.csv")

# Convert 'date' column to datetime format
train['date'] = pd.to_datetime(train['date'])
test['date'] = pd.to_datetime(test['date'])

# EDA and feature engineering
# Add extra features
train['year'] = train['date'].dt.year
train['month'] = train['date'].dt.month_name()
train['day'] = train['date'].dt.day_name()

test['year'] = test['date'].dt.year
test['month'] = test['date'].dt.month_name()
test['day'] = test['date'].dt.day_name()

# Add weekend feature
train['is_weekend'] = train['day'].apply(lambda x: 1 if x in ['Saturday', 'Sunday'] else 0)
test['is_weekend'] = test['day'].apply(lambda x: 1 if x in ['Saturday', 'Sunday'] else 0)

# Add time step feature
train['Time_step'] = np.arange(len(train))
test['Time_step'] = np.arange(len(test))

# Label encoding for categorical features
le = LabelEncoder()
cols = ['country', 'product', 'store', 'month', 'day', 'year']
for col in cols:
    train[col] = le.fit_transform(train[col])
    test[col] = le.transform(test[col])

# Train-validation split
X_train = train[train.date <= '2018-05-31'].drop(['date', 'num_sold'], axis=1)
X_val = train[(train.date >= '2018-06-01') & (train.date <= '2018-12-31')].drop(['date',
'num_sold'], axis=1)
y_train = train[train.date <= '2018-05-31']['num_sold']
y_val = train[(train.date >= '2018-06-01') & (train.date <= '2018-12-31')]['num_sold']
```

```python

# Model - Random Forest Regressor
baseline_regressor = RandomForestRegressor(n_estimators=500, n_jobs=-1)
baseline_regressor.fit(X_train, y_train)

# Validation predictions
val_pred = baseline_regressor.predict(X_val)

# SMAPE evaluation function
def SMAPE(y_true, y_pred):
    denominator = (y_true + np.abs(y_pred)) / 200.0
    diff = np.abs(y_true - y_pred) / denominator
    diff[denominator == 0] = 0.0
    return np.mean(diff)

# Calculate SMAPE
print('The SMAPE value of the Random Forests model is:', SMAPE(y_val, val_pred))

# Feature importance
importance = baseline_regressor.feature_importances_
feature_names = list(X_train.columns)
std = np.std([tree.feature_importances_ for tree in baseline_regressor.estimators_], axis=0)
baseline_importances = pd.Series(importance, index=feature_names)

# Plot feature importance
fig, ax = plt.subplots()
baseline_importances.plot.bar(xerr=std, ax=ax)
ax.set_title("Feature Importances")
ax.set_ylabel("Mean decrease in impurity")
fig.tight_layout()

# Predictions for test data
test_pred = baseline_regressor.predict(test.drop(['date'], axis=1))

# Prepare submission
sample['num_sold'] = test_pred
sample.to_csv("baseliner.csv", index=False)

# Visualization of predicted sales trends
test['num_sold'] = test_pred
data = pd.concat([train, test])

data_plot = data.groupby(['date']).mean().reset_index()
plt.figure(figsize=(15, 7))
sns.lineplot(x=data_plot.date, y=data_plot.num_sold)
plt.title('Number Sold Over Time')
plt.show()
import pandas as pd
import numpy as np
from statsmodels.tsa.arima.model import ARIMA
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error

```

```python
# Load Dataset
# Replace 'sales_data.csv' with your dataset file.
data = pd.read_csv('sales_data.csv', parse_dates=['Date'], index_col='Date')

# Data Preprocessing
data = data.asfreq('D')  # Ensure daily frequency
data.fillna(method='ffill', inplace=True)  # Fill missing values

# Visualization (optional)
import matplotlib.pyplot as plt
data['Sales'].plot(title="Sales Over Time")
plt.show()

# --- ARIMA Model ---
# Train-Test Split for Time Series
train_size = int(len(data) * 0.8)
train, test = data[:train_size], data[train_size:]

# Fit ARIMA Model
arima_model = ARIMA(train['Sales'], order=(5, 1, 0))  # Change parameters as needed
arima_fit = arima_model.fit()

# Forecast
arima_forecast = arima_fit.forecast(steps=len(test))
arima_results = pd.DataFrame({'Actual': test['Sales'], 'Forecast': arima_forecast})
print(arima_results)

# Evaluation Metrics for ARIMA
arima_mae = mean_absolute_error(test['Sales'], arima_forecast)
arima_rmse = np.sqrt(mean_squared_error(test['Sales'], arima_forecast))
print(f"ARIMA - MAE: {arima_mae}, RMSE: {arima_rmse}")

# --- Random Forest Regressor ---
# Feature Engineering
data['Day'] = data.index.day
data['Month'] = data.index.month
data['Year'] = data.index.year
data['Lag1'] = data['Sales'].shift(1)
data.dropna(inplace=True)

X = data[['Day', 'Month', 'Year', 'Lag1']]
y = data['Sales']

# Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Fit Random Forest Model
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Forecast
rf_forecast = rf_model.predict(X_test)

# Evaluation Metrics for Random Forest
```

```python
rf_mae = mean_absolute_error(y_test, rf_forecast)
rf_rmse = np.sqrt(mean_squared_error(y_test, rf_forecast))
print(f"Random Forest - MAE: {rf_mae}, RMSE: {rf_rmse}")

# Visualization of Forecasts
plt.figure(figsize=(12, 6))
plt.plot(test.index, test['Sales'], label="Actual Sales")
plt.plot(test.index, arima_forecast, label="ARIMA Forecast")
plt.plot(y_test.index, rf_forecast, label="RF Forecast", linestyle='dashed')
plt.legend()
plt.title("Sales Forecasting")
plt.show()
```