Univerzitet u Sarajevu
Elektrotehnički fakultet
Odsjek za računarstvo i informatiku

Zadaća 1

Vremenske serije

Dubinska analiza podataka

Profesor:

V. prof. dr. Dženana Đonko

Studenti:

Imamović Nasiha
Novalić Neira
Rovčanin Nejra
Rudalija Ema

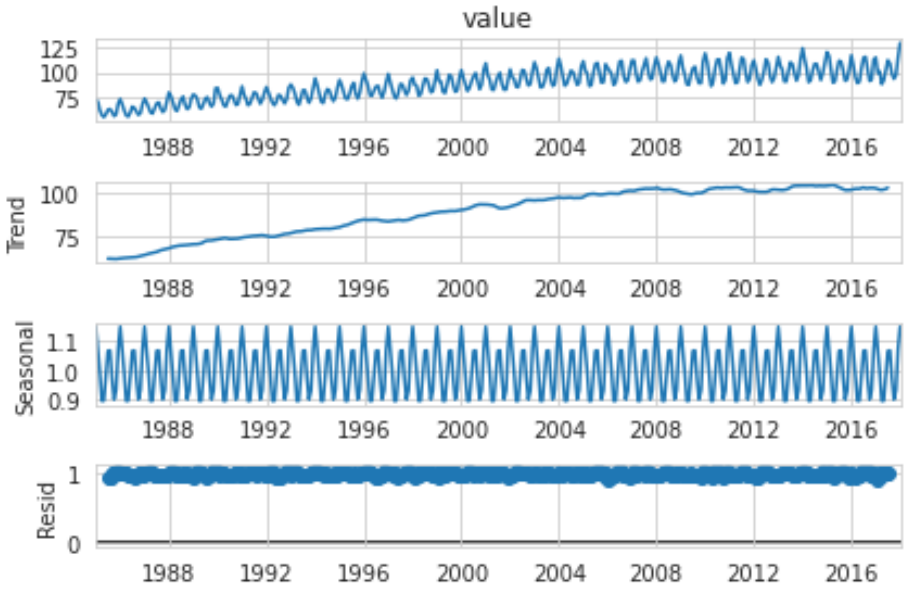
Sarajevo, mart 2023.

1. Uvod

1.1 Opis odabranog seta podataka

Skup podataka koji je korišten u ovoj zadaći je *Electric Production* preuzet sa linka: https://www.kaggle.com/datasets/shenba/time-series-datasets?resource=download.

Dataset sadrži vrijednosti količine proizvedene električne energije u Sjedinjenim Američkim Državama po svakom danu od 1985. do 2018. godine. Skup podataka se sastoji od dvije kolone - datum i vrijednost proizvodnje tipa *float*. Također, nisu pronađene NA vrijednosti. U ovoj vremenskoj seriji prisutni su trend i sezonalnost što se može vidjeti sa grafika ispod.



1.2 Definicija problema

Opisana vremenska serija je iskorištena za formiranje ARIMA i SARIMA kandidatskih modela od kojih su odabrani najbolji za oba predstavnika, na osnovu AIC

mjere kvaliteta. Nakon toga, dataset je podijeljen na trening i test podatke, a potom je urađena evaluacija odabranih modela.

U drugom dijelu zadaće obađeni su modeli Exponential Smoothing, Facebook Prophet, LSTM neuralne mreže i Amazon Deep AR algoritmi. Za istrenirane modele je urađena odgovarajuća evaluacija.

Podjela zadataka po članovima tima je predstavljena sljedećom tabelom:

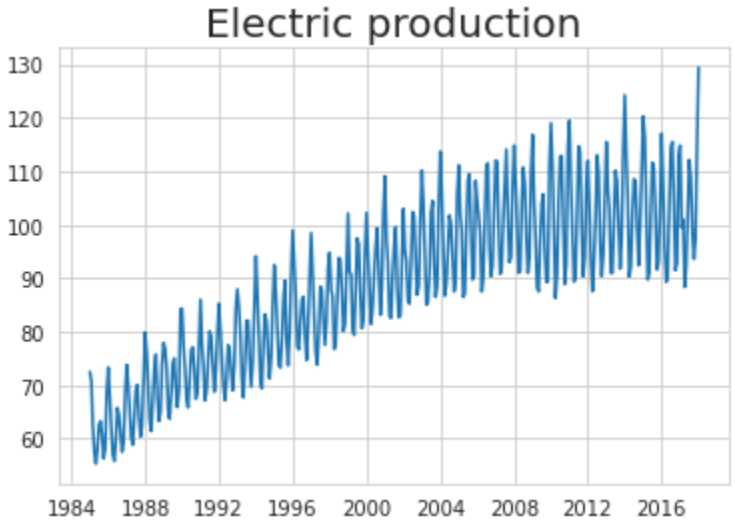
Zadatak	**Članovi**
ARIMA	Rovčanin Nejra i Rudalija Ema
SARIMA	Imamović Nasiha i Novalić Neira
Exponential Smoothing	Novalić Neira
Facebook Prophet	Rovčanin Neira
LSTM neuralne mreže	Imamović Nasiha
Amazon Deep AR	Rudalija Ema

2. ARIMA I SARIMA modeli

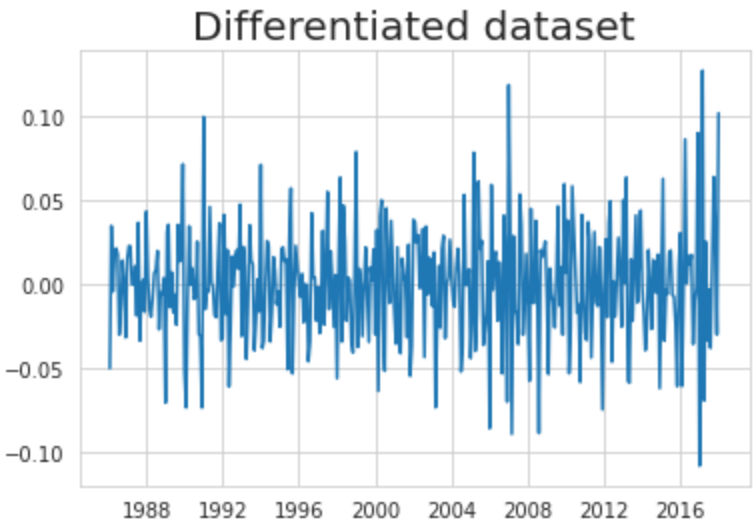
2.1 ARIMA

U ovom dijelu će biti opisani koraci analize i modeliranja ARIMA kandidatskih modela za odabranu vremensku seriju. Prvo smo izvršili analizu podobnosti podataka za modeliranje ARIMA modelom. Korišten je Ljung-Box test, a na osnovu rezultata je utvrđeno da među instancama postoji autokorelacija što znači da je vremenska serija pogodna za modeliranje ARIMA modelom.

Za provjeru stacionarnosti koristili smo ADF i KPSS testove. Prilikom prvog mjerenja, oba testa su pala pa je bilo jasno da serija nije stacionarna. Da bismo prošli ove testove, morali smo ukloniti pojavu trenda i sezonalnosti.

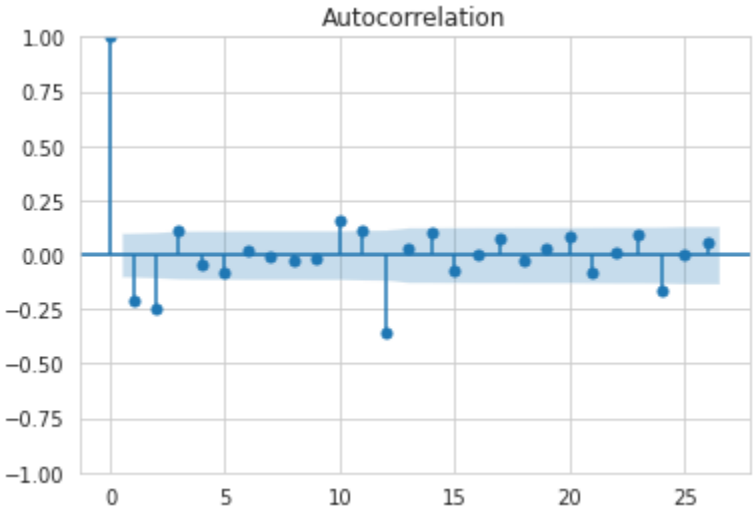


Prije uklanjanja pojave trenda i sezonalnosti, uočili smo značajne promjene u varijansi pa je urađena log-transformacija nad inicijalnim podacima. Nakon diferenciranja vremenske serije, uklonili smo trend i sezonalnost i na taj način pripremili set podataka za modeliranje.



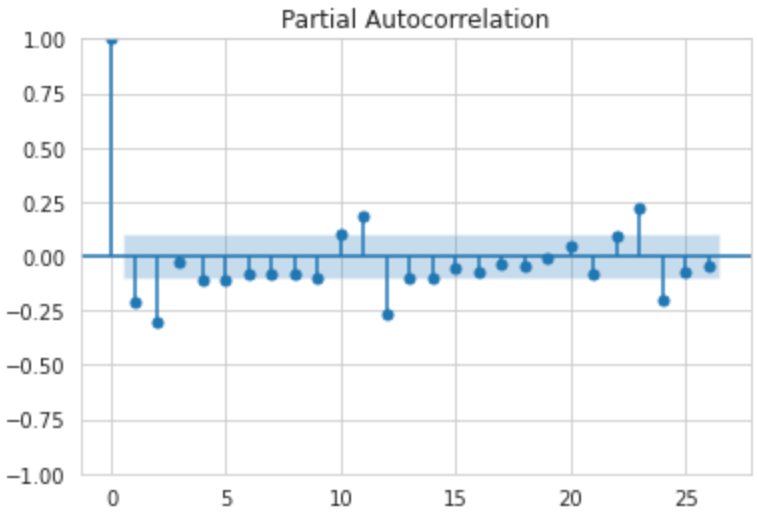
Nakon ovih izmjena, konačno smo prošli ADF i KPSS testove.

Grafik autokorelacije prikazan je na slici ispod.



Ovaj grafik nam je pomogao da odredimo parametar q, koji se odnosi na red moving average dijela procesa (MA). Sa grafika vidimo da se ističu vrijednosti u koracima k = 1, k = 2 i k = 3. Odstupanje u koraku br.12 pripisali smo slučajnosti.

Grafik parcijalne autokorelacije pomogao nam je u određivanju reda autoregresivnog dijela procesa (AR).



Očigledno je da se ističu vrijednosti na prva dva koraka pa smo za parametar p ispitali vrijednosti k = 1 i k = 2.

Budući da je diferenciranje urađeno jednom, parametar d je postavljen na 1.

Konačno, funkciji ARIMA šaljemo sve opisane parametre na način kako je prikazano u kodu ispod:

```
from statsmodels.tsa.arima.model import ARIMA

for p in [0,1,2]:
  for q in [0,1,2,3]:
    model = ARIMA(data_log, order=(p,1,q))
    model_fit = model.fit()
    display(model_fit.summary())
```

U nastavku su prikazani rezultati, odnosno vrijednosti AIC vrijednosti za dobivene kandidatske modele. Model koji ima najmanju AIC vrijednost je ARIMA(2,1,2) pa se upravo on uzima u dalje razmatranje prilikom analize evaluacije modela.

Kandidatski model	AIC mjera kvalitete
ARIMA (0,1,0)	-834.374
ARIMA (0,1,1)	-971.619

ARIMA (0,1,2)	-1047.241
ARIMA (0,1,3)	-1129.117
ARIMA (1,1,0)	-895.970
ARIMA (1,1,1)	-970.928
ARIMA (1,1,2)	-1085.921
ARIMA (1,1,3)	-1130.094
ARIMA (2,1,0)	-1143.379
ARIMA (2,1,1)	-1341.770
ARIMA (2,1,2)	-1499.606
ARIMA (2,1,3)	-1486.046

2.2 SARIMA

Proces uklanjanja trenda i sezonalnosti kod SARIMA modela je urađen na isti način kako je opisano u prethodnom poglavlju. Za određivanje sezonalnih parametara koristili smo funkciju *auto_arima* iz biblioteke *pmdarima*.

```
results=pm.auto_arima(data_log, d=1, start_p=1, start_q=1, max_p=2, max_q=3,
                      seasonal=True, m=7, information_criterion='aic',
trace=True, error_action='ignore', stepwise=True)
```

Parametri d, p i q su postavljeni na isti način kao i u ARIMA modelu. Parametar *seasonal=True* označava da je u pitanju vremenska serija koja je sezonalna, a *m* je parametar koji označava da su podaci serije na dnevnoj bazi.

Nakon treniranja modela sa svim kombinacijama, dobili smo da je model sa najmanjom AIC vrijednošću ARIMA(2,1,0)(2,0,2,7), što znači da su vrijednosti parametara:

- Red autoregresivnog dijela procesa: p = 2
- Red diferenciranja: d = 1
- Red moving average dijela procesa: q = 0
- Red autoregresivnog sezonalnog dijela procesa: P = 2
- Red sezonalnog diferenciranja: D = 0
- Red moving average sezonalnog dijela procesa: Q = 2

```
Performing stepwise search to minimize aic
 ARIMA(1,1,1)(1,0,1)[7] intercept   : AIC=-1022.185, Time=1.95 sec
 ARIMA(0,1,0)(0,0,0)[7] intercept   : AIC=-832.493, Time=0.14 sec
 ARIMA(1,1,0)(1,0,0)[7] intercept   : AIC=-947.514, Time=0.47 sec
 ARIMA(0,1,1)(0,0,1)[7] intercept   : AIC=-1024.783, Time=0.67 sec
 ARIMA(0,1,0)(0,0,0)[7]             : AIC=-834.374, Time=0.10 sec
 ARIMA(0,1,1)(0,0,0)[7] intercept   : AIC=-969.694, Time=0.18 sec
 ARIMA(0,1,1)(1,0,1)[7] intercept   : AIC=-1024.035, Time=0.84 sec
 ARIMA(0,1,1)(0,0,2)[7] intercept   : AIC=-1025.477, Time=1.47 sec
 ARIMA(0,1,1)(1,0,2)[7] intercept   : AIC=-1077.896, Time=1.73 sec
 ARIMA(0,1,1)(2,0,2)[7] intercept   : AIC=inf, Time=2.83 sec
 ARIMA(0,1,1)(2,0,1)[7] intercept   : AIC=-1270.860, Time=2.13 sec
 ARIMA(0,1,1)(2,0,0)[7] intercept   : AIC=-1136.733, Time=0.96 sec
 ARIMA(0,1,1)(1,0,0)[7] intercept   : AIC=-1006.016, Time=0.34 sec
 ARIMA(0,1,0)(2,0,1)[7] intercept   : AIC=-1269.021, Time=0.97 sec
 ARIMA(1,1,1)(2,0,1)[7] intercept   : AIC=-1273.857, Time=1.54 sec
 ARIMA(1,1,1)(2,0,0)[7] intercept   : AIC=-1135.278, Time=1.29 sec
 ARIMA(1,1,1)(2,0,2)[7] intercept   : AIC=-1355.105, Time=2.05 sec
 ARIMA(1,1,1)(1,0,2)[7] intercept   : AIC=-1075.806, Time=1.91 sec
 ARIMA(1,1,0)(2,0,2)[7] intercept   : AIC=inf, Time=2.79 sec
 ARIMA(2,1,1)(2,0,2)[7] intercept   : AIC=-1360.421, Time=2.72 sec
 ARIMA(2,1,1)(1,0,2)[7] intercept   : AIC=-1354.580, Time=2.02 sec
 ARIMA(2,1,1)(2,0,1)[7] intercept   : AIC=-1345.024, Time=1.98 sec
 ARIMA(2,1,1)(1,0,1)[7] intercept   : AIC=-1350.518, Time=1.17 sec
 ARIMA(2,1,0)(2,0,2)[7] intercept   : AIC=-1375.022, Time=2.08 sec
 ARIMA(2,1,0)(1,0,2)[7] intercept   : AIC=-1152.824, Time=2.22 sec
 ARIMA(2,1,0)(2,0,1)[7] intercept   : AIC=-1260.124, Time=2.81 sec
 ARIMA(2,1,0)(1,0,1)[7] intercept   : AIC=-1144.902, Time=0.84 sec
 ARIMA(2,1,0)(2,0,2)[7]             : AIC=inf, Time=1.71 sec

Best model:  ARIMA(2,1,0)(2,0,2)[7] intercept
Total fit time: 42.010 seconds
```

2.3 Evaluacija modela

U prethodnim poglavljima smo opisali način preprocesiranja podataka vremenske serije i odabir kandidatskih modela za ARIMA i SARIMA procese slijedeći vrijednosti AIC mjere kvaliteta. U ovom dijelu ćemo pokazati stvarnu primjenu izabrana dva modela.

Podatke iz dataseta smo podijelili na trening i testni skup tako da je testni, odnosno skup predviđen za prognoziranje (forecast) iznosio 20% ukupnih podataka.

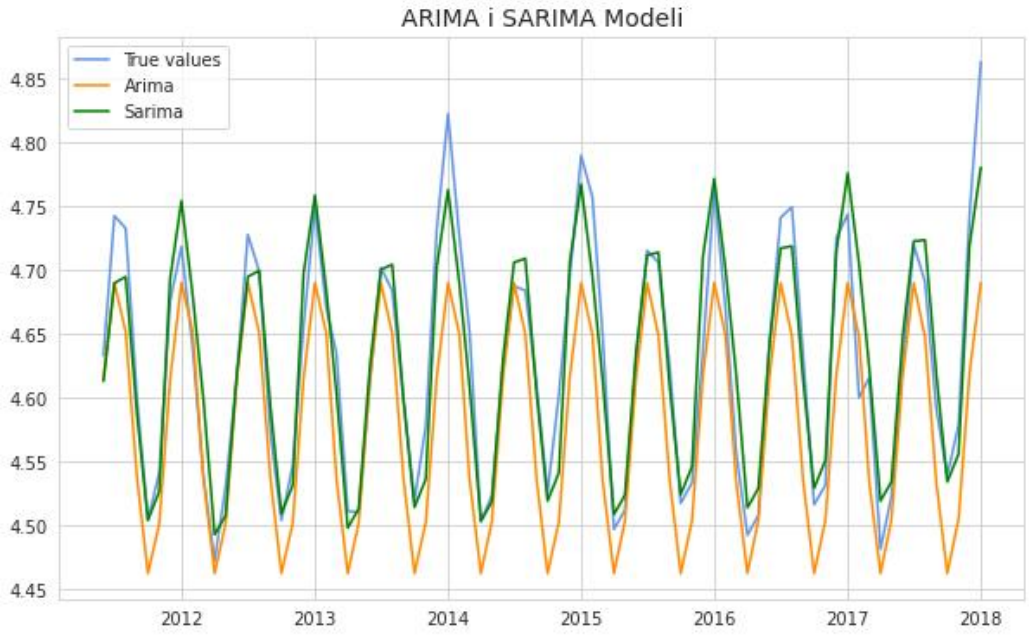
Treniranje modela nad trening skupom i evaluacija nad testnim skupom su prikazani u nastavku.

```
model1 = ARIMA(train, order=(2,1,2))
ARIMA_model = model1.fit()

predictions_arima = ARIMA_model.predict(start=len(train),
end=len(train)+len(test)-1, dynamic=False)
```

```
model2 = SARIMAX(train, order=(2, 1, 0), seasonal_order=(2,0,2,7))
SARIMA_model = model2.fit()
predictions_sarima = SARIMA_model.predict(start=len(train),
end=len(train)+len(test)-1, dynamic=False)
```

```
plt.figure(figsize = (10,6))
plt.plot(test, label = "True values", color = "cornflowerblue")
plt.plot(predictions_arima,label = "Arima", color='darkorange')
plt.plot(predictions_sarima, label="Sarima", color='green')
plt.title("ARIMA i SARIMA Modeli", size = 14)
plt.legend(loc = 'upper left')
plt.show()
```

Na narednom grafiku možemo vidjeti uspješnost oba modela. I ARIMA i SARIMA odabrani modeli su bili poprilično uspješni, s tim da je SARIMA model bio nešto precizniji, odnosno bliži stvarnim vrijednostima.



Modele smo evaluirali i tako što smo izračunali MSE, MAE i RMSE greške. U tabeli ispod vmožemo primijetiti da je SARIMA model zaista nešto bolji od ARIMA modela, što odgovara prethodnim tvrdnjama i zaključcima izvedenih iz grafičkog prikaza.

Model	MSE	MAE	RMSE
ARIMA	0.0039	0.051	0.06
SARIMA	0.001	0.023	0.03

3. Ostali algoritmi

3.1 Exponential Smoothing

Exponential Smoothing je metoda koja se koristi za rad sa vremenskim serijama. Ovaj algoritam radi tako što dodjeljuje eksponencijalno opadajuću težinu za prethodna opažanja, a naziv je dobio upravo po tome što se pomenuta težina dodijeljena svakom promatranju eksponencijalno smanjuje. Model također pretpostavlja da će posmatrane vrijednosti u budućnosti biti donekle isti ili slični sa vrijednostima iz bliske prošlosti. Ovaj pristup se generalno koristi za predviđanje podataka vremenske serije na osnovu prethodnih pretpostavki korisnika, kao što je sezonalnost ili sistematski trend. Inače, postoje tri vrste ovog algoritma:

- **Single Exponential Smoothing** - koristi se za podatke kod kojih nema ni trenda ni sezonalnosti
- **Double Exponential Smoothing** - koristi se za podatke kod kojih postoji pojava trenda, ali nema sezonalnosti
- **Triple Exponential Smoothing** - koristi se za podatke kod kojih je prisutan i trend i sezonalnost

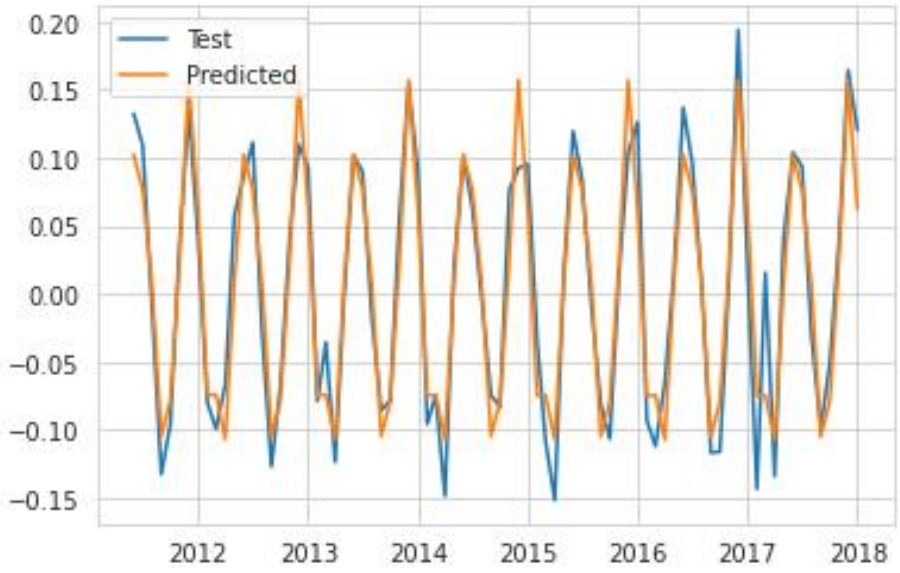
Posljednja metoda se još naziva i **Holt-Winters metoda** i upravo je ona korištena u našoj zadaći, budući da u podacima imamo i trend i sezonalnost.

Implementacija u Pythonu je vrlo jednostavna. Nakon podjele podataka na trening i testni skup, u svrhe treniranja je korištena funkcija *ExponentialSmoothing* iz paketa

statsmodel.tsa.api. Potom je pokrenuto treniranje, a onda je implementirano i predviđanje uz pomoć funkcije *forecast*.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.api import ExponentialSmoothing
from sklearn.metrics import mean_squared_error

# Učitavanje podataka
data = pd.read_csv("Electric_Production.csv", names=col_names, header=0,
parse_dates=[0])
data['date'] = pd.to_datetime(data['date'],infer_datetime_format=True)
data = data.set_index(['date'])

# Log transformacija vremenske serije
data_log = data
data_log['value']= np.log(data_log['value'])

# Podjela podataka na skupove za treniranje i testiranje
train_data = data.iloc[:int(len(data)*0.8)]
test_data = data.iloc[int(len(data)*0.8):]

# Definiranje modela i treniranje na skupu za treniranje
model = ExponentialSmoothing(train_data, trend="add", seasonal="add",
seasonal_periods=12)
fit_model = model.fit()

# Prognoziranje vrijednosti za skup za testiranje
predictions = fit_model.forecast(len(test_data))

# Grafički prikaz rezultata
plt.plot(test_data.index, test_data, label="Test")
plt.plot(predictions.index, predictions, label="Predicted")
plt.legend()
plt.show()
```

Na grafiku ispod možemo vidjeti performanse ovog modela, za koji je MSE greška iznosila 0.001.



3.2 Facebook Prophet

Ovo je open-source algoritam za generisanje modela vremenskih serija koji koristi kombinaciju starijih i novih ideja. Posebno je dobar u modeljiranju vremenskih serija koje imaju pojavu sezonalnosti. U osnovi se sastoji od zbira tri funkcije i greške:

$$y(t) = g(t) + s(t) + h(t) + err$$

Funkcija rasta (g) modelira ukupni trend podataka. Stara ideja je zasnovana na znanjima o linearnim i logističkim funkcijama, dok je nova ideja ugrađena u Facebook Prophet to da trend rasta može biti prisutan u svim tačkama u podacima. Funkcija rasta može biti linearna, logistička ili flat.

Funkcija sezonalnosti (s) je zapravo Fourierov niz kao funkcija vremena. Svaki sinusni i kosinusni član niza se množi sa nekim koeficijentom. Krajnja suma može aproksimirati gotovo bilo koju krivulju ili u slučaju Propheta, sezonski, odnosno ciklični obrazac.

Funkcija događaja (h) omogućava Prophetu da prilagodi predviđanje za vrijeme nekog praznika ili velikog događaja koji može promijeniti prognozu. Potrebna je lista datuma događaja te se na temelju historijskih podataka na utvrđene datume dodaje odnosno oduzima neka definisana vrijednost. Također, osim jednog datuma se može definisati i više dana oko nekog datuma, npr. dani prije i poslije Nove godine.

U nastavku je opisana implementacija ovog algoritma u Python programskom jeziku. Budući da algoritam zahtijeva da se nazivi kolona postave na *ds* i *y*, implementirali smo funkciju koja će to uraditi s našim podacima.

```
def preproc_prophet_data(df: pd.DataFrame) -> pd.DataFrame:
    data = df[["date", "value" ]]
    data.columns = ["ds", "y"]
    return data
```

Nakon toga, implementirana je funkcija *train_prophet_model* koja će nakon poziva kreirati *Prophet*-objekat jednostavnim pozivom funkcije *Prophet* iz istoimene biblioteke. Zatim se toj instanci, koju smo nazvali *model*, prosljeđuju preprocesirani trening podaci i konačno, funkcija vraća istrenirani model.

```
from prophet import Prophet

def train_prophet_model(train_data: pd.DataFrame)-> Prophet: ## Treniranje
    #Novi Prophet objekat
    model = Prophet()

    #Preprocesiranje podataka tako da zadovoljava uslove
    prophet_train_data = preproc_prophet_data(train_data)

    #Treniranje
```

```
    model.fit(prophet_train_data)

    return model
```

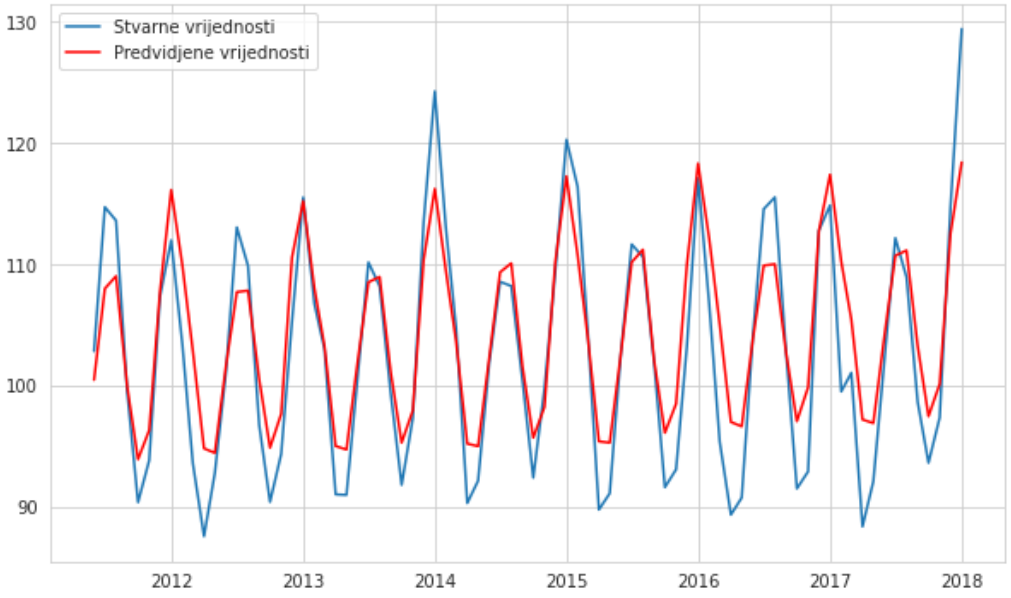
Naredna funkcija odnosi se na formiranje prognoza tako što se preprocesirani testni podaci šalju kao parametar funkciju *predict* pozvanom nad istreniranim modelom. Funkcija kao rezultat vraća predikcije.

```
def prophet_forecast(prophet_model: Prophet, test_data: pd.DataFrame) ->
pd.DataFrame:
    prophet_test_data = preproc_prophet_data(test_data)
    forecast = prophet_model.predict(prophet_test_data)
    return forecast
```

Konačno, sve opisane funkcije su primjenjene nad stvarnim podacima kao što je prikazano u kodu ispod.

```
# Učitavanje podataka
data = pd.read_csv("Electric_Production.csv", names=col_names, header=0,
parse_dates=[0])

# Podjela podataka na skupove za treniranje i testiranje
train_data = data.iloc[:int(len(data)*0.8)]
test_data = data.iloc[int(len(data)*0.8):]

#Formiranje modela
prophet_model = train_prophet_model(train_data)
prophet_model

#Predikcije
forecast = prophet_forecast(prophet_model, test_data)
forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail()
```

MSE-greška ovako istreniranog modela bila je 0.004, a uspješnost je prikazana na sljedećem grafu.



3.3 LSTM neuronske mreže

LSTM ili Long Short Term Memory su sekvencijalne neuronske mreže dubokog učenja koje omogućavaju perzistenciju informacija. Radi se o posebnoj vrsti rekurentne neuronske mrež (RNN) koja se može nositi sa problemom nestajanja gradijenta sa kojim se suočava klasični RNN. LSTM se u Python programskom jeziku može implementirati koristeći Keras biblioteku.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Dense, LSTM
from sklearn.metrics import mean_squared_error

# Učitavanje podataka
data = data_log_return
dataset = data.values
dataset = dataset.astype("float32")

# Podjela podataka na skupove za treniranje i testiranje
train_data = dataset[:int(len(dataset)*0.8)]
```

```
test_data = dataset[int(len(dataset)*0.8):]
```

Slično kao i kod prethodnog algoritma, podaci su podijeljeni na trening i testni skup. Ono što je bilo potrebno dodatno uraditi, a karakteristično je za LSTM neuronske mreže je prilagođavanje skupa podataka sa odgovarajućim vremenskim koracima. Prilagođavanje se može uraditi uz pomoć argumenta *lookup*, a služi za definiranje broja prethodnog vremena koji će se koristiti kao ulazna varijabla za predviđanje sljedećeg perioda.

Pošto smo još u prvom dijelu zadaće utvrdili da je red sezonalnog diferenciranja 12, argument *lookup* je postavljen upravo na tu vrijednost i kao takav proslijeđen funkciji *create_dataset*. Na taj način su formiranje varijable *train_X, train_Y, test_X i test_Y* sa preraspodijeljenim podacima.

```
#Funkcija za stvaranje skupa podataka s odgovarajućim vremenskim koracima
def create_dataset(dataset, look_back=1):
    dataX, dataY = [], []
    for i in range(len(dataset)-look_back-1):
        a = dataset[i:(i+look_back), 0]
        dataX.append(a)
        dataY.append(dataset[i + look_back, 0])
    return np.array(dataX), np.array(dataY)

# Stvaranje skupa podataka s odgovarajućim vremenskim koracima za treniranje i
testiranje
look_back = 12
train_X, train_Y = create_dataset(train_data, look_back)
test_X, test_Y = create_dataset(test_data, look_back)
```

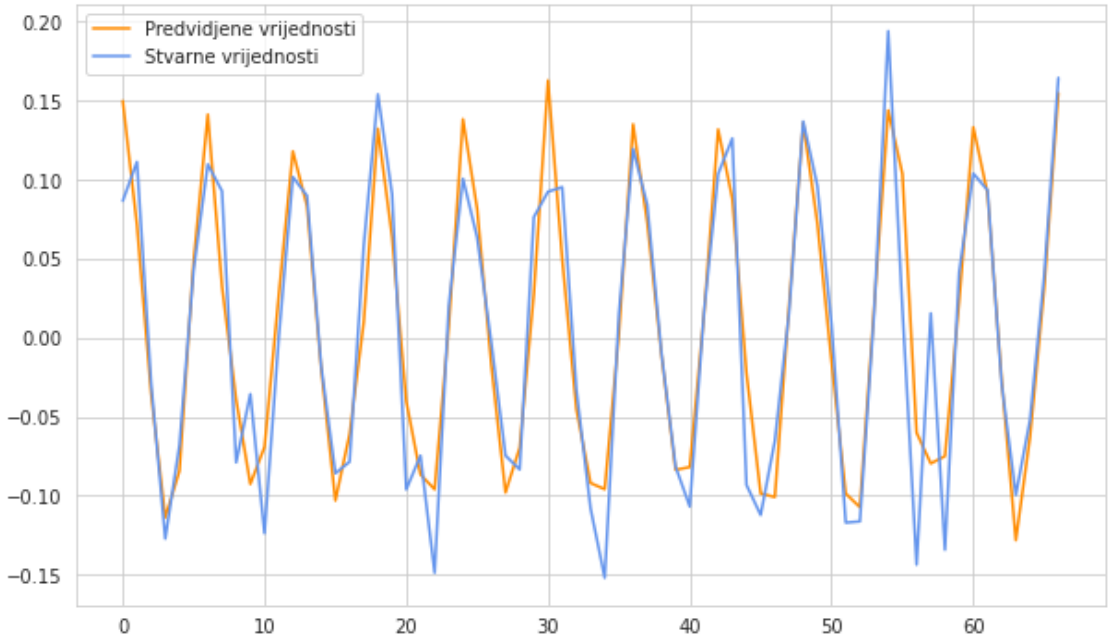
Nakon oblikovanja ulaznih podataka u format pogodan za LSTM neuronske mreže, slijedi definiranje modela i treniranje. Model je kreiran pozivom funkcije *Sequential*, a na njega su dodani jedan LSTM i jedan izlazni Dense sloj. Funkcija gubitka je MSE, a

optimizator Adam. Što se tiče treniranja, formirani model je istreniran u 100 epoha, a veličina batcha je 32.

Konačno, model je testiran nad testnim podacima uz pomoć funkcije *predict*.

```
# Oblikovanje ulaznih podataka za LSTM mrežu
train_X = np.reshape(train_X, (train_X.shape[0], 1, train_X.shape[1]))
test_X = np.reshape(test_X, (test_X.shape[0], 1, test_X.shape[1]))

# Definiranje modela i treniranje na skupu za treniranje
model = Sequential()
model.add(LSTM(4, input_shape=(1, look_back)))
model.add(Dense(1))
model.compile(loss="mean_squared_error", optimizer="adam")
model.fit(train_X, train_Y, epochs=100, batch_size=32, verbose=2)

# Prognoziranje vrijednosti za skup za testiranje
predictions = model.predict(test_X)
predictions = predictions.reshape(-1)
```

MSE greška za ovakav model iznosi 0.0013, a njegove performanse su prikazane na sljedećem grafu.



3.4 Amazon DeepAR

Amazon DeepAR algoritam za predviđanje je algoritam za supervizirano učenje za predviđanje jednodimenzionalnih vremenskih serija koristeći rekurentne neuronske mreže (RNN). Klasične metode predviđanja kao što su ARIMA ili Exponential Smoothing uklapaju jedan model u svaku pojedinačnu vremensku seriju. Zatim se taj model koristi za ekstrapolaciju vremenske serije u budućnosti. Ukoliko skup podataka sadrži stotine povezanih vremenskih serija, DeepAR nadmašuje standardne ARIMA, SARIMA i ETS metode.

Ulaz samog algoritma je jedna ili, po mogućnosti, više ciljnih vremenskih serija generiranih istim procesom. Na osnovu skupa ulaznih podataka algoritam trenira model koji uči aproksimaciju ovog procesa i koristi ga da predvidi kako će se ciljna serija razvijati.

Svaka ciljna vremenska serija može biti povezana sa vektorom statičkih, vremenski nezavisnih kategoričkih karakteristika i vektorom dinamičkih, vremenski zavisnih vremenskih serija.

Tokom treninga DeepAR prihvata skup podataka za trening i opcioni skup podataka za testiranje. Koristi skup testnih podataka za procjenu obučenog modela. Ovi skupovi se sastoje od jedne ili više ciljnih serija.

U nastavku je prikazan kod u Python programskom jeziku sa naznačenim implementacijskim detaljima u okviru komentara. Značajno je istaći da je za ovaj algoritam bilo potrebno pretvoriti podatke iz tipa *DataFrame* u *ListDataset*. Za potrebe implementacije korištene su gotove biblioteke *gluonts* i *mxnet*.

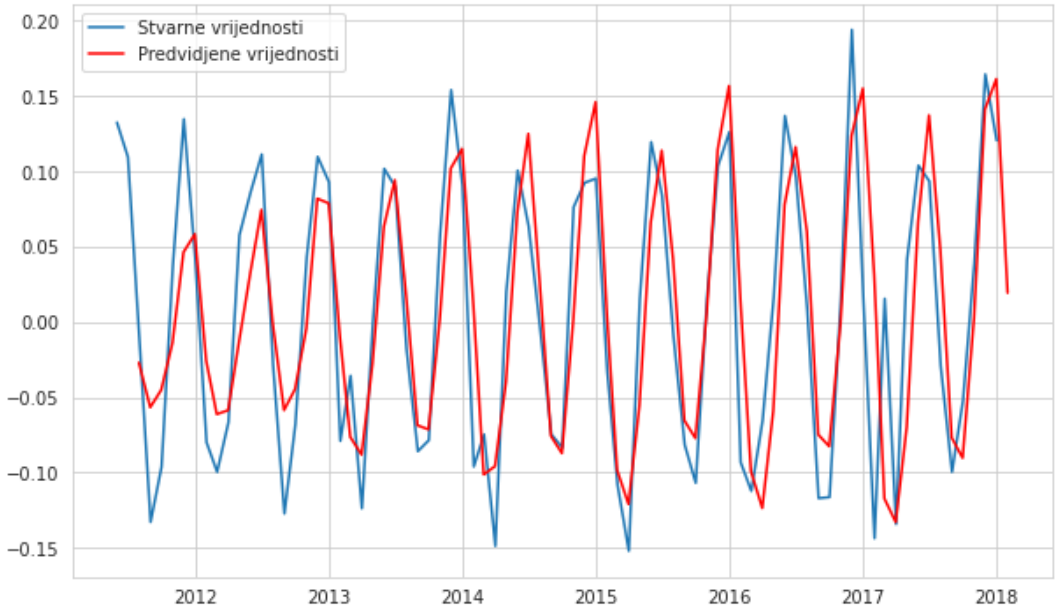
```
import pandas as pd
from gluonts.dataset.common import ListDataset
from gluonts.mx.trainer import Trainer
from gluonts.evaluation.backtest import make_evaluation_predictions
from gluonts.evaluation import Evaluator
import numpy as np
import matplotlib.pyplot as plt
from gluonts.dataset.common import ListDataset
from gluonts.mx.trainer import Trainer
from gluonts.evaluation.backtest import make_evaluation_predictions
from gluonts.mx.model.deepar import DeepAREstimator
from gluonts.dataset.common import ListDataset
import mxnet as mx


col_names = ["date", "value"]
# Učitavanje podataka
data = pd.read_csv("Electric_Production.csv", names=col_names, header=0,
parse_dates=[0])
data.columns = ['date', 'value']
data['date'] = data['date'].dt.date

data['date'] = data['date'].apply(date_parser)

data = data.set_index('date')

#log transformacija vremenske serije
data_log = data
data_log['value']= np.log(data_log['value'])
data = data_log.diff().iloc[1: , :]

train_data = data.iloc[:int(len(data)*0.8)]
test_data = data.iloc[int(len(data)*0.8):]
```

```
# Stvaranje formata podataka za GluonTS
training_data = ListDataset(
    [{"start": train_data.index[0], "target": train_data.value.values}],
    freq="M"
)

# Definiranje DeepAR estimatora i treniranje na skupu za treniranje
estimator = DeepAREstimator(
    freq="M",
```

```
    prediction_length= 79,
    trainer=Trainer(
        epochs=2,
        learning_rate=1e-3
    )
)

predictor = estimator.train(training_data=training_data)
```

```
# Prognoziranje vrijednosti za skup za testiranje
test_ds = ListDataset(
    [{"start": test_data.index[0], "target": test_data.value}],
    freq="M"
)

forecast_it, ts_it = make_evaluation_predictions(
    dataset=test_ds, predictor=predictor, num_samples=100
)
forecasts = list(forecast_it)
tss = list(ts_it)

# Evaluacija predikcija
evaluator = Evaluator(quantiles=[0.1, 0.5, 0.9])
agg_metrics, item_metrics = evaluator(iter(tss), iter(forecasts),
num_series=len(test_ds))

print(agg_metrics)
```

Na kraju, ovaj model je imao MSE grešku 0.0085, a njegove performanse su prikazane na grafu ispod.



4. Zaključak

U ovom dijelu ćemo pogledati greške i performanse svih korištenih modela. Od četiri izabrana modela iz drugog zadatka, najboljim se pokazao algoritam Exponential Smoothing čije su performanse slične performansama SARIMA-modela. SARIMA model ima neznatno bolju MAE vrijednost greške.

Deep AR, koji je češće namijenjen treniranju više vremenskih serija, pokazao se najlošijim za naš dataset.

Generalno, možemo reći da su se svi modeli dobro istrenirali nad našom vremeskom serijom. Razlog za to je činjenica što u našim podacima nije bilo nedostajućih vrijednosti, outliera i svega što bi zahtijevalo izvođenje koraka transformacije i čišćenja podataka.

Također, broj podataka je bio dovoljan da se opisani modeli istreniraju i pokažu dobre prognoze u budućnosti.

Model	MSE	MAE	RMSE
ARIMA	0.0039	0.051	0.06
SARIMA	0.001	0.023	0.03
Exponential Smoothing	0.001	0.025	0.03
FB Prophet	0.004	0.05	0.05
LSTM neuralne mreže	0.002	0.03	0.04
Deep AR	0.008	0.09	0.1

Napomene:

1) Zbog bolje preglednosti, odlučili smo se na prikazivanje grafova za svaki pojedinačni algoritam što je prikazano u prethodnim dijelovima izvještaja. Zbog različite pripreme i prilagođavanja podataka za svaki algoritam, nismo bili u mogućnosti iscrtati ih sve na jednom grafu.

2) Pri analizi dataseta i podešavanju istog za odabrane algoritme, vršili smo kombinovanje različitih parametara kako bi dobile najbolje razultate. Neki od parametara su:

- omjer trening i testnog skupa pri čemu se 80:20 pokazao kao najbolji
- podešavanje stope učenja pri čemu se 10 ^ -3 pokazala kao najefikasnija
- podešavanje epoha za učenje

Samo određivanje parametara za ARIMA i SARIMA model smo odradile preko pomoćnih funkcija te tu nismo imale problema da iz prvog pokušaja dobijemo odlične rezultate.