ELECTRICITY PRICES PREDICTION

TEAM MEMBER

LAVANYA T

Phase 3 Submission Document

Project : ELECTRICITY PRICES PREDICTION

Introduction:

In a world that thrives on energy as the lifeblood of modern society, the dynamics of electricity pricing have a profound impact on both consumers and producers. The ability to accurately predict electricity prices holds immense significance for various stakeholders, ranging from individual households seeking to manage their energy costs to utility companies striving to optimize resource allocation and policy makers working towards a sustainable energy future.

Electricity price prediction is not merely a matter of financial prudence; it's a critical element in the broader landscape of energy management and sustainability. It empowers us to make informed decisions, reduce energy waste, and align our consumption patterns with fluctuating supply and demand dynamics.

This discussion or project aims to delve into the intricate world of electricity price prediction. We will explore the multifaceted factors that influence pricing, from supply and demand patterns to environmental conditions and regulatory policies. Through the lens of data-driven approaches, machine learning, and statistical modeling, we will uncover the methodologies and tools used to forecast electricity prices with increasing precision.

Throughout our journey, we will address the real-world implications of electricity price prediction. From enabling cost-efficient strategies for businesses to encouraging renewable energy adoption and grid optimization, the ability to foresee price trends stands as a linchpin in the pursuit of an efficient, sustainable, and equitable energy ecosystem.

As we embark on this exploration of electricity price prediction, we invite you to discover the intricate interplay between data, technology, and the future of energy management. Join us as we uncover the valuable insights hidden within the numbers and explore the potential to make more informed, economically sound, and environmentally responsible decisions in an electrified world.

Data Source

A good data source for electricity prices prediction using machine learning should be Accurate, Complete & Accessible.

Dataset link: https://www.kaggle.com/datasets/chakradharmattapalli/electricity-price-prediction

1	DateTime	Holiday	HolidayFl	DayOfWe	WeekOfY	Day	Month	Year	PeriodOfD	ForecastW	SystemLo	SMPEA	ORKTemp	ORKWind	CO2Inten	ActualWir	SystemLo	SMPEP2
2	########	None	0	1	44	1	11	2011	0	315.31	3388.77	49.26	6	9.3	600.71	356	3159.6	54.32
3	########	None	0	1	44	1	11	2011	1	321.8	3196.66	49.26	6	11.1	605.42	317	2973.01	54.23
4	########	None	0	1	44	1	11	2011	2	328.57	3060.71	49.1	5	11.1	589.97	311	2834	54.23
5	########	None	0	1	44	1	11	2011	3	335.6	2945.56	48.04	6	9.3	585.94	313	2725.99	53.47
6	########	None	0	1	44	1	11	2011	4	342.9	2849.34	33.75	6	11.1	571.52	346	2655.64	39.87
7	########	None	0	1	44	1	11	2011	5	342.97	2810.01	33.75	5	11.1	562.61	342	2585.99	39.87
8	########	None	0	1	44	1	11	2011	6	343.18	2780.52	33.75	5	7.4	545.81	336	2561.7	39.87
9	########	None	0	1	44	1	11	2011	7	343.46	2762.67	33.75	5	9.3	539.38	338	2544.33	39.87
10	########	None	0	1	44	1	11	2011	8	343.88	2766.63	33.75	4	11.1	538.7	347	2549.02	39.87
11	########	None	0	1	44	1	11	2011	9	344.39	2786.8	33.75	4	7.4	540.39	338	2547.15	39.87
12	########	None	0	1	44	1	11	2011	10	345.02	2817.59	33.75	4	7.4	532.3	372	2584.58	39.87
13	########	None	0	1	44	1	11	2011	11	342.23	2895.62	47.42	5	5.6	547.57	361	2641.37	39.87
14	########	None	0	1	44	1	11	2011	12	339.22	3039.67	44.31	5	3.7	556.14	383	2842.19	51.45
15	########	None	0	1	44	1	11	2011	13	335.39	3325.1	45.14	5	3.7	590.34	358	3082.97	51.45
16	########	None	0	1	44	1	11	2011	14	330.95	3661.02	46.25	4	9.3	596.22	402	3372.55	52.82
17	########	None	0	1	44	1	11	2011	15	325.93	4030	52.84	5	3.7	581.52	368	3572.64	53.65
18	########	None	0	1	44	1	11	2011	16	320.91	4306.54	59.44	5	5.6	577.27	361	3852.42	54.21
19	########	None	0	1	44	1	11	2011	17	365.15	4438.05	62.15	6	5.6	568.76	340	4116.03	58.33
20	########	None	0	1	44	1	11	2011	18	410.55	4585.84	61.81	8	7.4	560.79	358	4345.42	58.33
21	########	None	0	1	44	1	11	2011	19	458.56	4723.93	61.88	9	7.4	542.8	339	4427.29	58.33
22	########	None	0	1	44	1	11	2011	20	513.17	4793.6	61.46	?	?	535.37	324	4460.41	58.33
23	########	None	0	1	44	1	11	2011	21	573.36	4829.44	61.28	11	13	532.52	335	4493.22	58.27

Necessary Steps to Follow

1. Data Collection

Gather historical electricity price data from a reliable source or database. This data can be in the form of a CSV file, API, or database query.

Program

```
import pandas as pd

# Load electricity price data from a CSV file
data = pd.read_csv('electricity_price_data.csv')
```

2. Data Preprocessing

Clean the data, handle missing values, and format it appropriately.

Program

```
# Remove rows with missing values
data.dropna(inplace=True)
```

3. Feature Selection/Engineering

Select relevant features and create new ones if needed.

Program

```
# Select features
selected_features = ['feature1', 'feature2', 'feature3']

# Create lag features for time series data
data['lag_price_1'] = data['electricity_price'].shift(1)
```

4. Data Splitting

Split the data into training, validation, and test sets.

Program

```
from sklearn.model_selection import train_test_split

X = data[selected_features]
y = data['electricity_price']

X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3,
random_state=42)
X_valid, X_test, y_valid, y_test = train_test_split(X_temp, y_temp, test_size=0.5,
random_state=42)
```

5.Model Selection and Training

Choose an appropriate prediction model and train it using the training data.

Program

```
from sklearn.linear_model import LinearRegression

# Initialize the model
model = LinearRegression()

# Train the model
model.fit(X_train, y_train)
```

6.Model Validation

Validate the model's performance on the validation set and fine-tune hyperparameters if necessary.

Program

```
# Validate the model
y_valid_pred = model.predict(X_valid)
```

```
  # Calculate evaluation metrics (e.g., RMSE)
from sklearn.metrics import mean_squared_error
  rmse = mean_squared_error(y_valid, y_valid_pred, squared=False)

  # Fine-tune hyperparameters and repeat validation if needed
```

7.Model Testing

Test the final model on the test set to assess its performance.

Program

```
# Test the final model
y_test_pred = model.predict(X_test)

# Calculate evaluation metrics (e.g., RMSE)
test_rmse = mean_squared_error(y_test, y_test_pred, squared=False)
```

8.Visualization

Create visualizations to understand how well the model captures electricity price trends.

Program

```
 import matplotlib.pyplot as plt

  # Plot actual vs. predicted prices
plt.plot(y_test, label='Actual Prices')
plt.plot(y_test_pred, label='Predicted Prices')
plt.legend()
  plt.title('Electricity Price Prediction')
plt.show()
```

9.Deployment

If the model performs well, deploy it to make real-time predictions or use it for scenario analysis.

10.Monitoring and Maintenance

Continuously monitor the model's performance and update it as new data becomes available.

Challenges Involved In Loading and Preprocessing A Electricity Prices Prediction

1.Data Quality and Completeness

Electricity price data may have missing values or inaccuracies, which need to be addressed through imputation or data cleansing techniques.

2.High-Dimensional Data

Electricity price prediction often involves a large number of features, including historical price data, weather information, and economic indicators. Handling high-dimensional data requires careful feature selection or dimensionality reduction.

3.Time Series Data

Electricity prices are typically time series data, which may exhibit seasonality, trends, and autocorrelation. Managing these temporal patterns and selecting appropriate time-based features can be challenging.

4.Data Volume

Energy market data can be voluminous, and loading and processing large datasets may require substantial computational resources and efficient data storage.

5.Data Frequency

Electricity price data may come at various time intervals (hourly, daily, etc.), and aligning data with a consistent time frame is necessary for analysis.

6.Normalization and Scaling

Properly normalizing or scaling data is crucial to ensure all features are on a common scale, but determining the right scaling method can be challenging.

7.Handling Categorical Data

Energy market data may include categorical variables such as geographical regions, power plants, or market types. Encoding and managing categorical data appropriately is essential.

8.Outliers and Anomalies

Electricity price data can be influenced by unusual events, like extreme weather conditions or market disruptions. Identifying and handling outliers and anomalies can be complex.

9.Feature Engineering

Creating meaningful lag features, seasonality indicators, or interaction terms requires domain expertise and an understanding of the factors affecting electricity prices.

10.Data Integration

Combining data from various sources, such as market reports, weather data, and economic indicators, can be challenging in terms of data alignment and consistency.

11.Model Selection

Choosing the right predictive model for electricity price forecasting is a challenge. It depends on the specific characteristics of the data and the desired prediction horizon.

12. Cross-Validation

Electricity price prediction models should be evaluated using appropriate crossvalidation techniques that account for the temporal nature of the data, which can be more complex than traditional cross-validation.

13. Real-Time Updates

In some cases, you may need to update your predictive model in real-time as new electricity price data becomes available. Ensuring the model remains accurate in such scenarios is challenging.

14. Regulatory and Market Dynamics

Electricity markets are subject to regulatory changes and market dynamics that can impact pricing. Incorporating such external factors into models can be complex.

15. Interpretable Models

Balancing model complexity with interpretability is challenging. Some stakeholders require transparent models to understand and trust predictions.

How To Overcome The Challenges Of Loading And Preprocessing Of A Electricity Prices Dataset

1.Data Quality Assurance

Carefully inspect the data for missing values and inconsistencies. Use data imputation techniques, such as mean imputation or interpolation, to handle missing data. Consider data validation against known benchmarks or expert domain knowledge.

2. High-Dimensional Data

Implement feature selection techniques to identify the most relevant features for prediction. Principal Component Analysis (PCA) and feature importance from tree-based models can help reduce dimensionality.

3. Time Series Data

Address seasonality and trends by differencing, detrending, or decomposing the time series data. Use lag features to capture temporal dependencies. Consider using models specifically designed for time series forecasting, such as ARIMA or LSTM.

4. Data Volume

Use efficient data storage solutions and distributed computing frameworks if you're dealing with extremely large datasets. Consider data aggregation or sampling to reduce the volume while retaining essential information.

5. Data Frequency

Resample the data to a consistent time frame that suits your analysis, whether it's hourly, daily, or some other interval.

6.Normalization and Scaling

Experiment with different scaling methods (e.g., Min-Max scaling or z-score normalization) to find the most suitable for your dataset and predictive model. Scaling should be applied consistently to all features.

7.Handling Categorical Data

Encode categorical variables using techniques like one-hot encoding, label encoding, or target encoding. Consider domain-specific knowledge when handling categorical data, especially if it represents geographic regions.

8.Outliers and Anomalies

Identify and handle outliers using statistical methods or domain expertise. Robust statistical techniques or anomaly detection algorithms can help in mitigating the impact of outliers.

9.Feature Engineering

Engage with domain experts to create meaningful lag features, seasonality indicators, and interactions. Experiment with different feature engineering techniques to capture important patterns.

10.Data Integration

Ensure consistency in data integration by aligning time frames and handling variations in data sources. Use data transformation and merging techniques to create a unified dataset.

11.Model Selection

Experiment with a range of prediction models and evaluate their performance. Consider the specific characteristics of your data, such as its seasonality and temporal dependencies, when choosing the appropriate model.

12. Cross-Validation

Implement time series cross-validation techniques like rolling-window or expanding-window cross-validation to assess model performance. Ensure that you validate the model's ability to generalize to unseen future data.

13. Real-Time Updates

If real-time updates are required, build a mechanism for updating your model with new data and continuously retraining it to maintain accuracy over time.

14. Regulatory and Market Dynamics

Stay informed about regulatory changes and market dynamics, and incorporate this information into your models or use it to adjust your predictions.

15. Interpretable Models

Balance model complexity with interpretability by using techniques like SHAP values or LIME to explain model predictions to stakeholders who require transparency.

Program:

ELECTRICITY PRICES PREDICTION

In[1]:

```
import pandas as pd
import numpy as np

from keras import callbacks
from keras import layers, models

from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_absolute_error

import xgboost as xgb
import tensorflow as tf

import matplotlib.pyplot as plt import
matplotlib.dates as mdates
```

In[2]:

df_total = pd.read_csv("/kaggle/input/energy-consumption-generationprices-and-weather/energy_dataset.csv", parse_dates=["time"], date_parser=lambda date: pd.to_datetime(date).tz_convert('Europe/Madrid'), index_col="time")

df = df_total[["total load actual", "price actual"]] df = df.rename(columns={"total load actual": "load", "price actual": "price"})
df

Out[2]:

	load	price
time		
2015-01-01 00:00:00+01:00	25385.0	65.41
2015-01-01 01:00:00+01:00	24382.0	64.92
2015-01-01 02:00:00+01:00	22734.0	64.48
2015-01-01 03:00:00+01:00	21286.0	59.32
2015-01-01 04:00:00+01:00	20264.0	56.04
...	...	...
2018-12-31 19:00:00+01:00	30653.0	77.02
2018-12-31 20:00:00+01:00	29735.0	76.16
2018-12-31 21:00:00+01:00	28071.0	74.30
2018-12-31 22:00:00+01:00	25801.0	69.89
2018-12-31 23:00:00+01:00	24455.0	69.88

In[3]:

time_column = df.index

```
min_datetime = time_column.min() max_datetime
= time_column.max()
hourly_datetimes = pd.date_range(min_datetime, max_datetime, freq="1H")

if np.sum(time_column == hourly_datetimes) == len(time_column):
    print("Times are correct.")
```

In[4]:

```
def fill_nans(df, column_name):

dfc = df[column_name]
nans_num = np.sum(np.isnan(dfc))

dfc = dfc.interpolate(method='linear', axis=0).ffill().bfill()
df[column_name] = dfc

print(f"Interpolated {nans_num} NaNs in column '{column_name}'.")

fill_nans(df, "price")
fill_nans(df, "load")
```

```
Interpolated 0 NaNs in column 'price'.
Interpolated 36 NaNs in column 'load'.
```

In[5]:

```
time_normalization = {
    "hour": lambda time: time / 24,
    "dayofweek": lambda time: time / 7,
    "month": lambda time: (time - 1) / 12,
```

```
}
datetimes = time_column.to_series().dt
 for time_property, normalize_time_func in
time_normalization.items():
    time = getattr(datetimes, time_property)
time_normalized = normalize_time_func(time)
df[time_property] = time_normalized
```

```
df
```

Out[5]:

	load	price	hour	dayofweek	month
time					
2015-01-01 00:00:00+01:00	25385.0	65.41	0.000000	0.428571	0.000000
2015-01-01 01:00:00+01:00	24382.0	64.92	0.041667	0.428571	0.000000
2015-01-01 02:00:00+01:00	22734.0	64.48	0.083333	0.428571	0.000000
2015-01-01 03:00:00+01:00	21286.0	59.32	0.125000	0.428571	0.000000
2015-01-01 04:00:00+01:00	20264.0	56.04	0.166667	0.428571	0.000000
...	...	...	...	...	...
2018-12-31 19:00:00+01:00	30653.0	77.02	0.791667	0.000000	0.916667
2018-12-31 20:00:00+01:00	29735.0	76.16	0.833333	0.000000	0.916667
2018-12-31 21:00:00+01:00	28071.0	74.30	0.875000	0.000000	0.916667
2018-12-31 22:00:00+01:00	25801.0	69.89	0.916667	0.000000	0.916667
2018-12-31 23:00:00+01:00	24455.0	69.88	0.958333	0.000000	0.916667

In[6]:

past_time_stamps = 24 target_time
= 24

```
total_time_window = past_time_stamps + target_time

# 2016 is a leap year train_split
= (365 + 366) * 24
val_split = train_split + 365 * 24

X_attr = df[["load", "price"]].to_numpy()
# time properties are already normalized
X_time_norm = df[["hour", "dayofweek", "month"]].to_numpy()
Y_price = df[["price"]].to_numpy()
Y_load = df[["load"]].to_numpy()

# X will be shifted later
# shift Y values 48h ahead
X_attr_train = X_attr[:train_split]
X_time_train_norm = X_time_norm[:train_split]
Y_load_train = Y_load[total_time_window:train_split]
Y_price_train = Y_price[total_time_window:train_split]

X_attr_val = X_attr[train_split:val_split]
X_time_val_norm = X_time_norm[train_split:val_split]
Y_load_val = Y_load[train_split+total_time_window:val_split]
Y_price_val = Y_price[train_split+total_time_window:val_split]

X_attr_test = X_attr[val_split:]
X_time_test_norm = X_time_norm[val_split:]
Y_load_test = Y_load[val_split+total_time_window:]
Y_price_test = Y_price[val_split+total_time_window:]

X_scaler = MinMaxScaler(feature_range=(0, 1))
X_scaler.fit(X_attr_train)
X_attr_train_norm = X_scaler.transform(X_attr_train)
X_attr_val_norm = X_scaler.transform(X_attr_val)
X_attr_test_norm = X_scaler.transform(X_attr_test)
```

```
Y_load_scaler = MinMaxScaler(feature_range=(0, 1))
Y_load_scaler.fit(Y_load_train)
Y_load_train_norm = Y_load_scaler.transform(Y_load_train)
Y_load_val_norm = Y_load_scaler.transform(Y_load_val)
Y_load_test_norm = Y_load_scaler.transform(Y_load_test)

Y_price_scaler = MinMaxScaler(feature_range=(0, 1))
Y_price_scaler.fit(Y_price_train)
Y_price_train_norm = Y_price_scaler.transform(Y_price_train)
Y_price_val_norm = Y_price_scaler.transform(Y_price_val)
Y_price_test_norm = Y_price_scaler.transform(Y_price_test)

def to_time_windows(X):
time_windows = []

   # use past measurements between 0..23h
for i in range(past_time_stamps):
       time_windows.append(X[i:-total_time_window+i, :])

   time_windows = np.stack(time_windows, axis=1)
return time_windows

# those will be used with the baseline models
X_attr_time_train_norm = np.concatenate([X_attr_train_norm,
X_time_train_norm], axis=1)
X_attr_time_val_norm = np.concatenate([X_attr_val_norm,
X_time_val_norm], axis=1)
X_attr_time_test_norm = np.concatenate([X_attr_test_norm,
X_time_test_norm], axis=1)

X_attr_time_train_norm_windows =
to_time_windows(X_attr_time_train_norm)
X_attr_time_val_norm_windows =
to_time_windows(X_attr_time_val_norm) X_attr_time_test_norm_windows
=
to_time_windows(X_attr_time_test_norm)
```

 # those will be used with our model

X_attr_train_norm_windows = to_time_windows(X_attr_train_norm)

X_attr_val_norm_windows = to_time_windows(X_attr_val_norm)

X_attr_test_norm_windows = to_time_windows(X_attr_test_norm)

X_time_train_norm_values = X_time_train_norm[past_time_stamps:target_time]

X_time_val_norm_values = X_time_val_norm[past_time_stamps:target_time]

X_time_test_norm_values = X_time_test_norm[past_time_stamps:target_time]

print("Train data", X_attr_time_train_norm_windows.shape, X_attr_train_norm_windows.shape, X_time_train_norm_values.shape, Y_load_train_norm.shape, Y_price_train_norm.shape)

print("Validation data", X_attr_time_val_norm_windows.shape, X_attr_val_norm_windows.shape, X_time_val_norm_values.shape, Y_load_val_norm.shape, Y_price_val_norm.shape)

print("Test data", X_attr_time_test_norm_windows.shape, X_attr_test_norm_windows.shape, X_time_test_norm_values.shape, Y_load_test_norm.shape, Y_price_test_norm.shape)

Out[6]:

```
Train data (17496, 24, 5) (17496, 24, 2) (17496, 3) (17496, 1) (17496, 1)
Validation data (8712, 24, 5) (8712, 24, 2) (8712, 3) (8712, 1) (8712, 1)
Test data (8712, 24, 5) (8712, 24, 2) (8712, 3) (8712, 1) (8712, 1)
```

In[7]:

n_features = X_attr_time_train_norm_windows.shape[2]

 # source: https://www.kaggle.com/code/varanr/hourly-energy-demandtime-series-forecast/ def lstm_model():

```
    model = models.Sequential(name="LSTM")
    model.add(layers.LSTM(25, input_shape=(past_time_stamps, n_features)))
    model.add(layers.Dropout(0.2))
    model.add(layers.Dense(1))

    return model
```

 # source: https://www.kaggle.com/code/nicholasjhana/multi-variatetime-series-forecasting-tensorflow def cnn_model():

```
    model = models.Sequential([
        layers.Conv1D(64, kernel_size=6, activation='relu',
input_shape=(past_time_stamps, n_features)),
        layers.MaxPooling1D(2),         layers.Conv1D(64,
kernel_size=3, activation='relu'),
layers.MaxPooling1D(2),         layers.Flatten(),
layers.Dropout(0.3),         layers.Dense(128),
layers.Dropout(0.3),         layers.Dense(1)
    ], name="CNN")

    return model
```

 # source: https://www.kaggle.com/code/dimitriosroussis/electricityprice-forecasting-with-dnns-eda

```
def run_xgboost(column_to_predict, x_train_norm, y_train_norm,
x_val_norm, y_val_norm, x_test_norm, y_test_norm, y_scaler, y_val, y_test):
    x_train_xgb = x_train_norm.reshape(-1, x_train_norm.shape[1] *
x_train_norm.shape[2])
    x_val_xgb = x_val_norm.reshape(-1, x_val_norm.shape[1] *
x_val_norm.shape[2])
    x_test_xgb = x_test_norm.reshape(-1, x_test_norm.shape[1] *
x_test_norm.shape[2])

    param = {'eta': 0.03, 'max_depth': 180,
'subsample': 1.0, 'colsample_bytree': 0.95,
            'alpha': 0.1, 'lambda': 0.15, 'gamma': 0.1,
```

```
        'objective': 'reg:linear', 'eval_metric': 'mae',
        'silent': 1, 'min_child_weight': 0.1, 'n_jobs': -1}

  dtrain = xgb.DMatrix(x_train_xgb, y_train_norm)
dval = xgb.DMatrix(x_val_xgb, y_val_norm)
  dtest = xgb.DMatrix(x_test_xgb, y_test_norm)
eval_list = [(dtrain, 'train'), (dval, 'val')]     evals_result = {}
  xgb_model = xgb.train(param, dtrain, 180, eval_list,
evals_result=evals_result, verbose_eval=False)

  y_val_pred_norm = xgb_model.predict(dval)
y_val_pred_norm = y_val_pred_norm.reshape(-1, 1)
y_val_pred = y_scaler.inverse_transform(y_val_pred_norm)
mae_val = mean_absolute_error(y_val, y_val_pred)
  print('XGBoost - MAE on validation dataset', round(mae_val, 2))

  y_test_pred_norm = xgb_model.predict(dtest)
y_test_pred_norm = y_test_pred_norm.reshape(-1, 1)
y_test_pred = y_scaler.inverse_transform(y_test_pred_norm)
mae_test = mean_absolute_error(y_test, y_test_pred)
print('XGBoost - MAE on test dataset', round(mae_test, 2))

  history = {
    "loss": evals_result['train']['mae'],
    "val_loss": evals_result['val']['mae']
  }

  plot_loss(history)
  plot_prediction(column_to_predict, y_val, y_val_pred)

  lstm_load_model = lstm_model() lstm_price_model
= lstm_model()
lstm_price_model.summary()

print("\n\n")
cnn_load_model = cnn_model() cnn_price_model
= cnn_model()
```

cnn_price_model.summary()

Out[7]:

```
Model: "LSTM" _________________________________________________________________
Layer (type)                 Output Shape              Param #
================================================================= lstm_11
(LSTM)                 (None, 25)                3100
_________________________________________________________________
dropout_31 (Dropout)         (None, 25)                0
_________________________________________________________________
dense_31 (Dense)             (None, 1)                 26
=================================================================
Total params: 3,126
Trainable params: 3,126
Non-trainable params: 0
_________________________________________________________________


Model: "CNN"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_22 (Conv1D)           (None, 19, 64)            1984
_________________________________________________________________
max_pooling1d_22 (MaxPooling (None, 9, 64)             0
_________________________________________________________________
conv1d_23 (Conv1D)           (None, 7, 64)             12352
_________________________________________________________________
max_pooling1d_23 (MaxPooling (None, 3, 64)             0
_________________________________________________________________
flatten_11 (Flatten)         (None, 192)               0
_________________________________________________________________
dropout_34 (Dropout)         (None, 192)               0
_________________________________________________________________ dense_34
(Dense)                (None, 128)               24704
_________________________________________________________________
dropout_35 (Dropout)         (None, 128)               0
_________________________________________________________________ dense_35
(Dense)                (None, 1)                 129
=================================================================
Total params: 39,169
Trainable params: 39,169
Non-trainable params: 0
_________________________________________________________________
```

In[8]:

```
def gru_encoder(time_series):
    x = layers.GRU(units=16)(time_series)

    return [x, "GRU"]


def cnn_encoder(time_series):
    num_filters = 8

    x = layers.Conv1D(num_filters * 1, 3, activation="relu",
padding="same")(time_series)
    x = layers.Conv1D(num_filters * 2, 3, activation="relu", padding="same",
strides=2)(x)
    x = layers.Conv1D(num_filters * 2, 3, activation="relu",
padding="same")(x)
    x = layers.Conv1D(num_filters * 3, 3, activation="relu",
padding="same", strides=2)(x)
    x = layers.Conv1D(num_filters * 3, 3, activation="relu", padding="same")(x)
    x = layers.Conv1D(num_filters * 4, 3, activation="relu",
padding="same", strides=2)(x)
    x = layers.Conv1D(num_filters * 4, 3, activation="relu",
padding="same")(x)
    x = layers.Flatten()(x)

    return [x, "CNN"]


def conv_encoder(time_series):
    x = tf.transpose(time_series, [0, 2, 1])
    x = layers.Conv1D(16, 3, padding="same", activation="linear")(x)
x = layers.Conv1D(8, 3, padding="same", activation="linear")(x)     x
= layers.Conv1D(4, 3, padding="same", activation="linear")(x)     x =
layers.Flatten()(x)
```

```
    return [x, "CONV"]


def concat_model(feature_encoder):     # past 24
timestamps of load and price     time_windows =
layers.Input((past_time_stamps, 2))     # hour, day of
the week, month of the time stamp when the forecast
happens
    time_values = layers.Input((3))

    x, encoder_name = feature_encoder(time_windows)
y = layers.Dense(16, activation="relu")(time_values)
    x = layers.Concatenate()([x, y])

    x = layers.Dense(16, activation="relu")(x)
x = layers.Dropout(rate=0.1)(x)     x =
layers.Dense(1)(x)

    model = models.Model(inputs = [time_windows, time_values], outputs=x,
                  name=f"{encoder_name}_concat_time")

    return model

gru_concat_time_load_model = concat_model(gru_encoder)
gru_concat_time_price_model = concat_model(gru_encoder)
gru_concat_time_price_model.summary()

print("\n\n")
cnn_concat_time_load_model = concat_model(cnn_encoder)
cnn_concat_time_price_model = concat_model(cnn_encoder)
cnn_concat_time_price_model.summary()

print("\n\n")
conv_concat_time_load_model = concat_model(conv_encoder)
conv_concat_time_price_model = concat_model(conv_encoder)
conv_concat_time_price_model.summary()
```

In[9]:

```
def plot_loss(history):

plt.plot(history["loss"])
plt.plot(history["val_loss"])
plt.title('Model loss')
plt.ylabel('loss') plt.xlabel('epoch')
plt.legend(['training', 'validation'], loc='upper right') plt.show()

two_weeks = 24 * 14
x =
time_column.to_series().to_numpy()[train_split+total_time_window:trai
n_split+total_time_window+two_weeks]

def plot_prediction(column_to_predict, y_true, y_pred):

plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%d.%m.%Y
%H:%M', tz=time_column.tz))
   plt.title(f"Electricity {column_to_predict}")
plt.plot(x, y_true[:two_weeks])     plt.plot(x,
y_pred[:two_weeks])
   plt.legend(['ground truth', 'prediction'], loc='upper left')
plt.gcf().autofmt_xdate()
   plt.show()
```

In[10]:

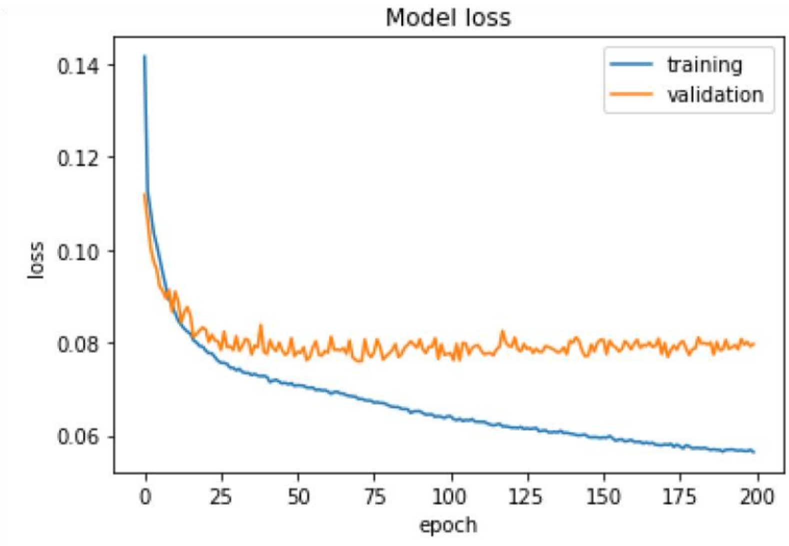
```
epochs = 200
# 0 - no output, 1 - detailed output, 2 - brief output verbosity
= 0
```

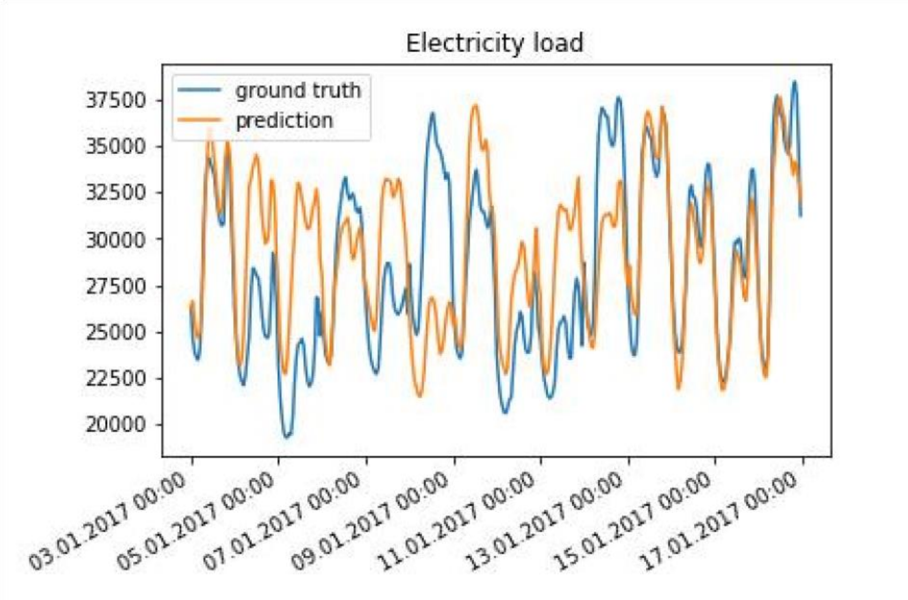
```
def run_model(model, column_to_predict, x_train_norm, y_train_norm,
x_val_norm, y_val_norm, x_test_norm, y_scaler, y_val, y_test):
   weights_path = f"{model.name}_{column_to_predict}_weights.h5"
modelckpt_callback = callbacks.ModelCheckpoint(
       monitor="val_loss",
filepath=weights_path,        verbose=verbosity,
save_weights_only=True,
       save_best_only=True,
   )

   model.compile(loss='mae', optimizer="adam")

   history = model.fit(x_train_norm, y_train_norm, epochs=epochs,
            validation_data=(x_val_norm, y_val_norm),
callbacks=[modelckpt_callback],             verbose=verbosity)

   model.load_weights(weights_path)

   y_val_pred_norm = model.predict(x_val_norm)
   y_val_pred = y_scaler.inverse_transform(y_val_pred_norm)

   mae_val = mean_absolute_error(y_val, y_val_pred)
   print(f"{model.name} - MAE on validation dataset", round(mae_val,
2))
   y_test_pred_norm = model.predict(x_test_norm)
   y_test_pred = y_scaler.inverse_transform(y_test_pred_norm)

   mae_test = mean_absolute_error(y_test, y_test_pred)
   print(f"{model.name} - MAE on test dataset", round(mae_test, 2))

   plot_loss(history.history)
   plot_prediction(column_to_predict, y_val, y_val_pred)
```

In[11]:

```
run_model(lstm_load_model, "load",
        X_attr_time_train_norm_windows, Y_load_train_norm,
        X_attr_time_val_norm_windows, Y_load_val_norm,
        X_attr_time_test_norm_windows, Y_load_scaler, Y_load_val,
Y_load_test)
```

Out[11]:

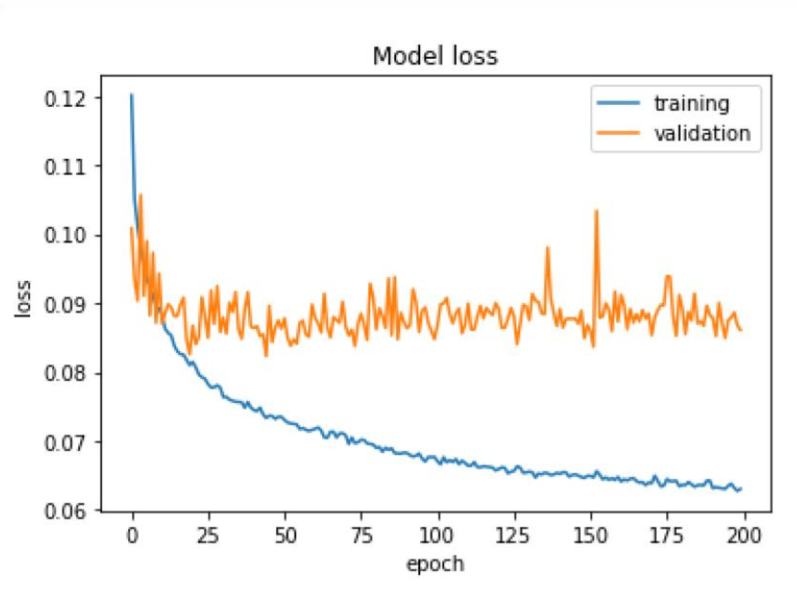


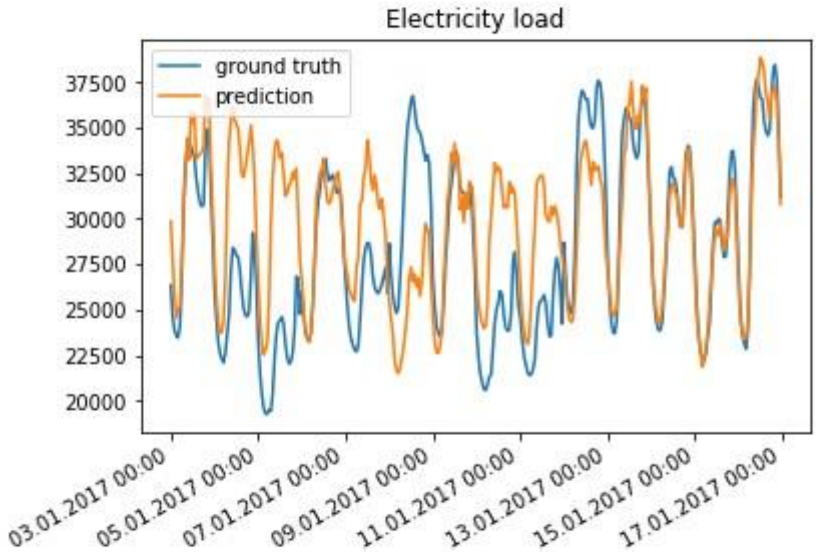


In[12]:

```
run_model(cnn_load_model, "load",
    X_attr_time_train_norm_windows, Y_load_train_norm,
    X_attr_time_val_norm_windows, Y_load_val_norm,
    X_attr_time_test_norm_windows, Y_load_scaler,
Y_load_val, Y_load_test)
```

Out[12]:



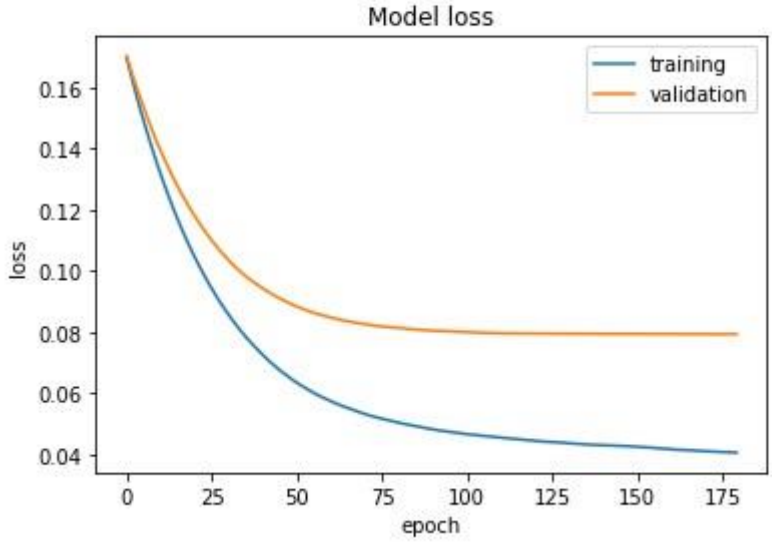


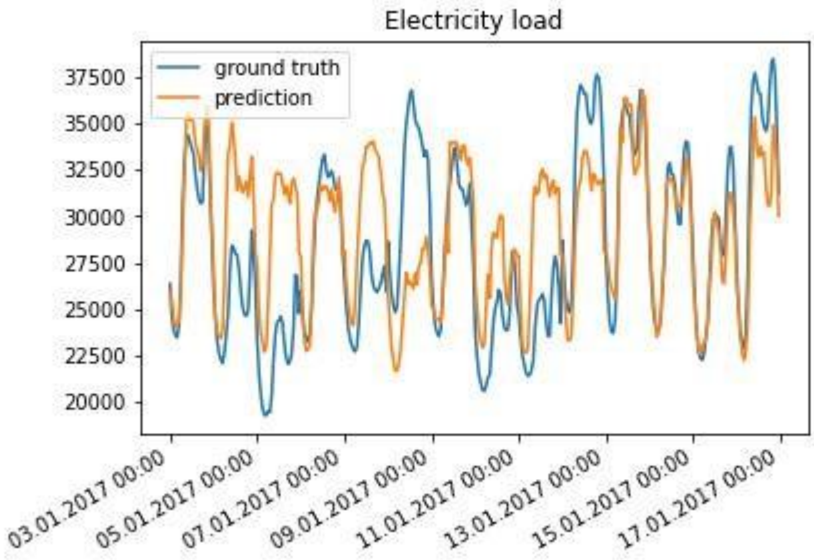
In[13]:

```
run_xgboost("load",
     X_attr_time_train_norm_windows,
Y_load_train_norm,
```

```
    X_attr_time_val_norm_windows, Y_load_val_norm,
X_attr_time_test_norm_windows, Y_load_test_norm,
    Y_load_scaler, Y_load_val, Y_load_test)
```

Out[13]:

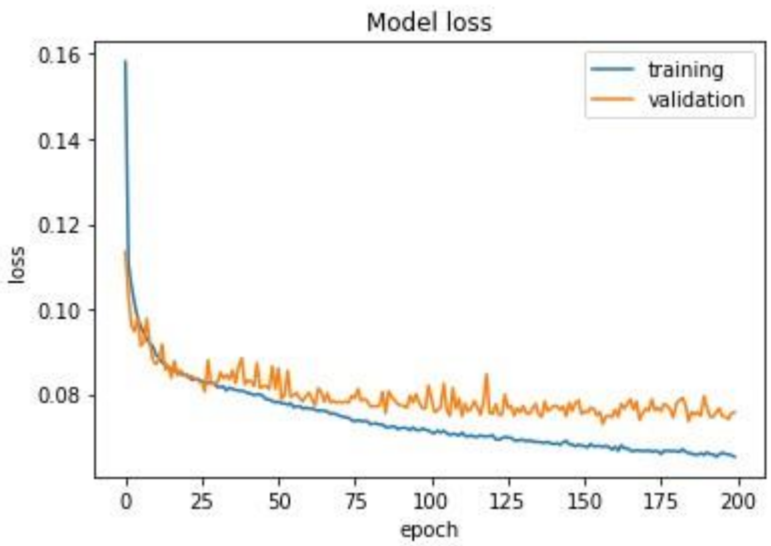


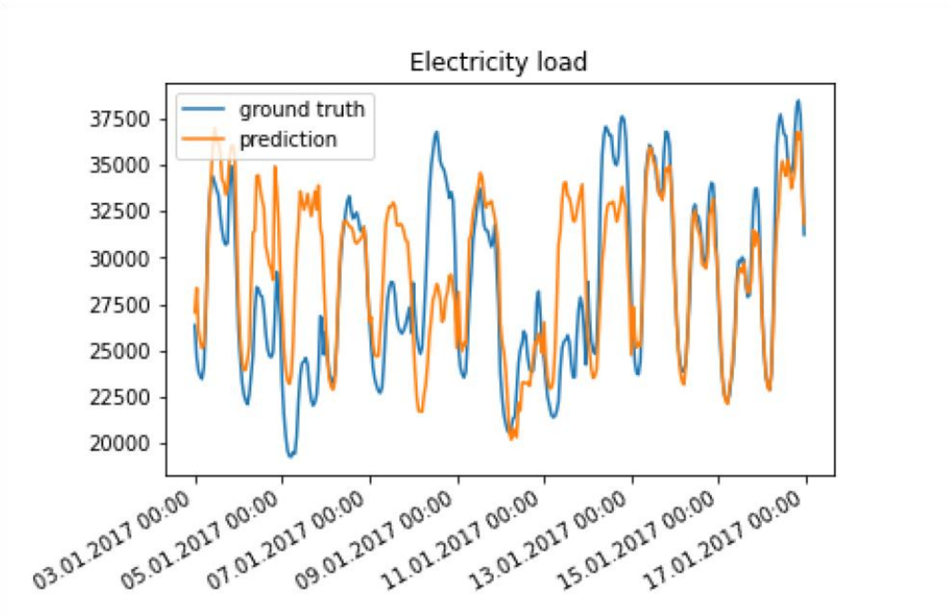


In[14]:

```
run_model(gru_concat_time_load_model, "load",
    [X_attr_train_norm_windows,
X_time_train_norm_values], Y_load_train_norm,
    [X_attr_val_norm_windows, X_time_val_norm_values],
Y_load_val_norm,
    [X_attr_test_norm_windows,
X_time_test_norm_values], Y_load_scaler, Y_load_val,
Y_load_test)
```

Out[14]:

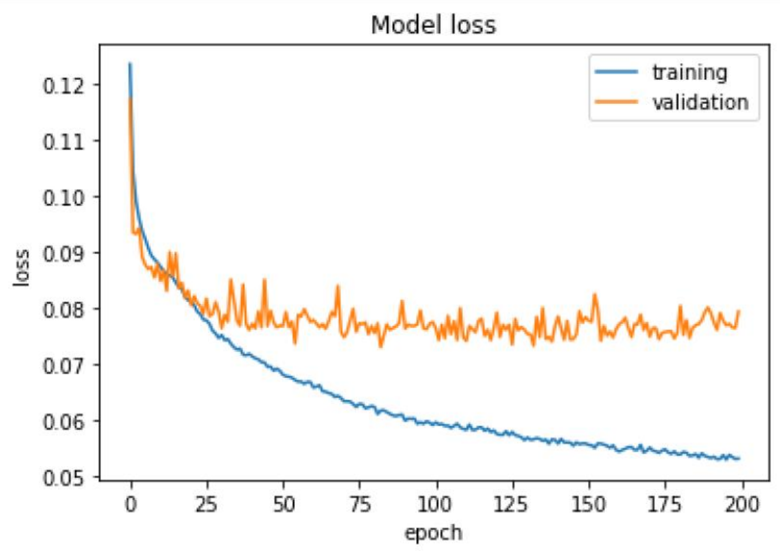


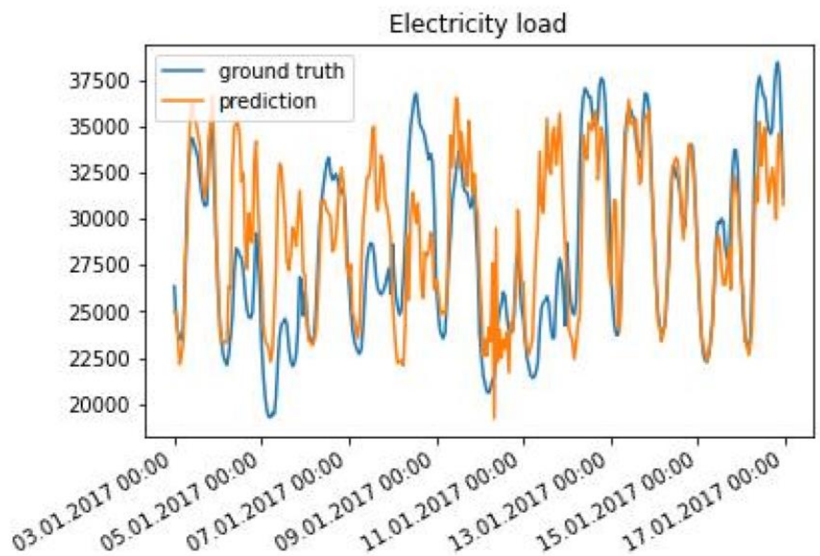


In[15]:

```
run_model(cnn_concat_time_load_model, "load",
     [X_attr_train_norm_windows,
X_time_train_norm_values], Y_load_train_norm,
[X_attr_val_norm_windows,
X_time_val_norm_values], Y_load_val_norm,
     [X_attr_test_norm_windows,
X_time_test_norm_values], Y_load_scaler, Y_load_val,
Y_load_test)
```

Out[15]:



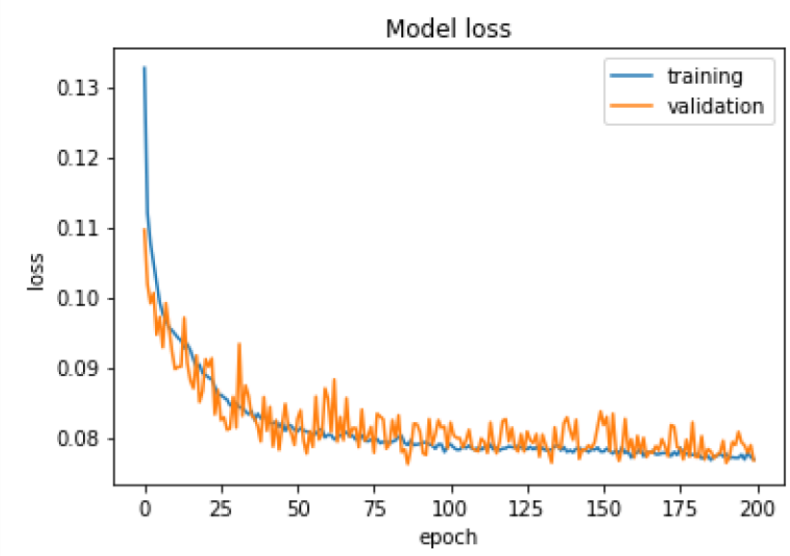


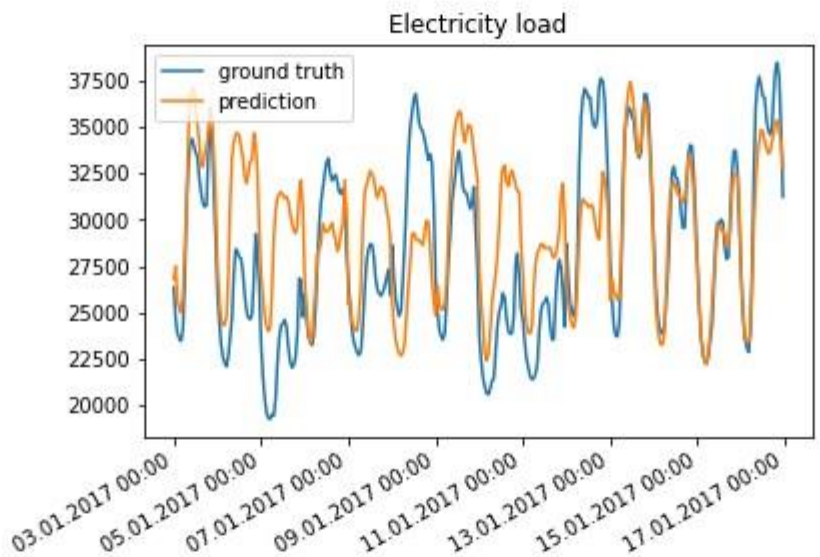
In[16]:

```
run_model(conv_concat_time_load_model, "load",
     [X_attr_train_norm_windows,
X_time_train_norm_values], Y_load_train_norm,
[X_attr_val_norm_windows,
X_time_val_norm_values], Y_load_val_norm,
     [X_attr_test_norm_windows,
X_time_test_norm_values], Y_load_scaler, Y_load_val,
```

```
Y_load_test)
```

Out[16]:



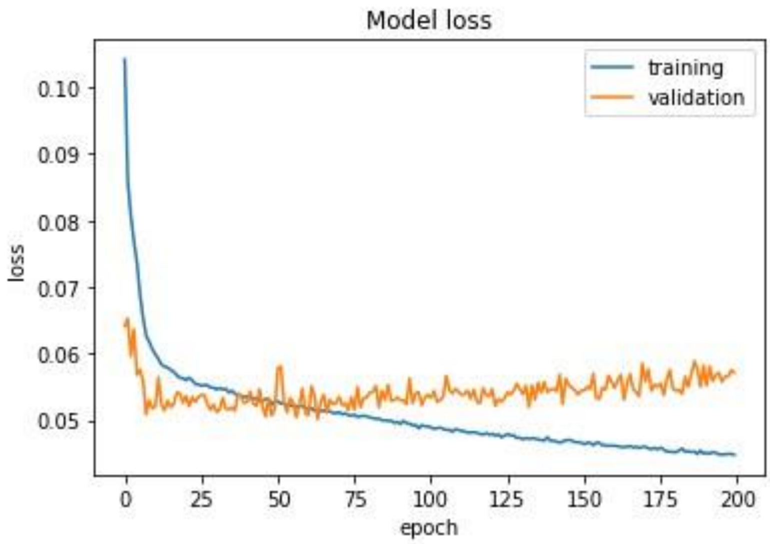


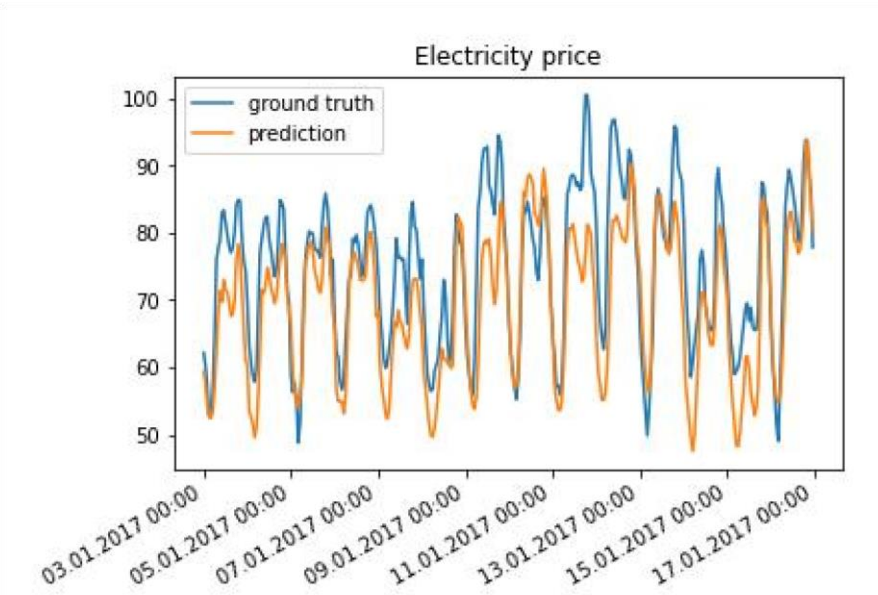
In[17]:

```
run_model(lstm_price_model, "price",
     X_attr_time_train_norm_windows,
Y_price_train_norm,
```

```
        X_attr_time_val_norm_windows, Y_price_val_norm,
        X_attr_time_test_norm_windows, Y_price_scaler,
Y_price_val, Y_price_test)
```

Out[17]:

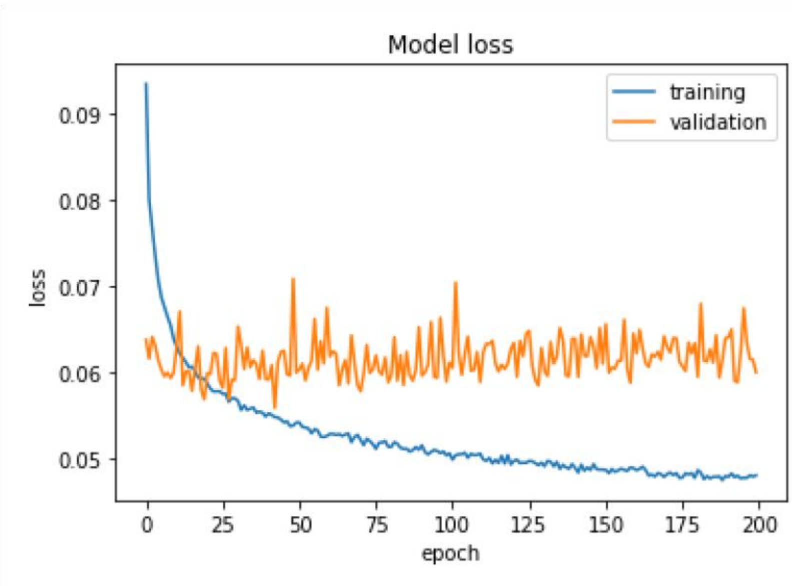


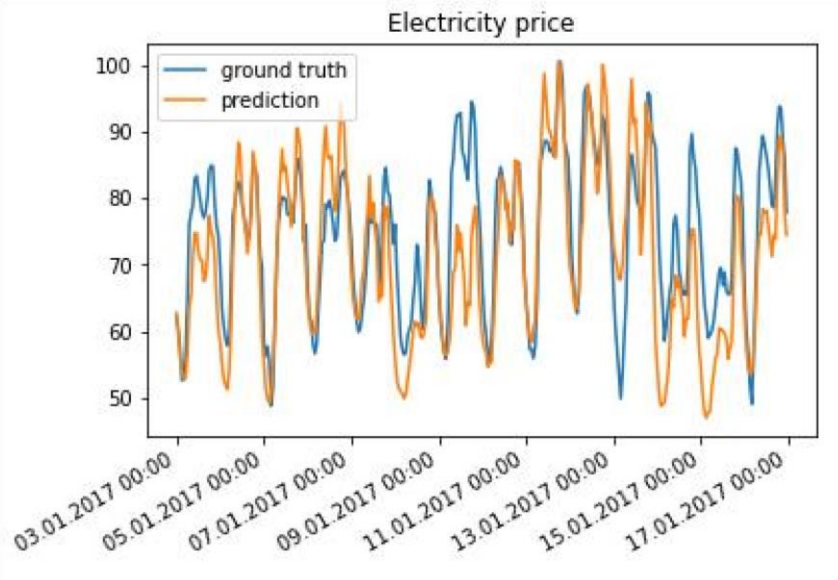


In[18]:

```
run_model(cnn_price_model, "price",
     X_attr_time_train_norm_windows,
Y_price_train_norm,
     X_attr_time_val_norm_windows, Y_price_val_norm,
     X_attr_time_test_norm_windows, Y_price_scaler,
Y_price_val, Y_price_test)
```

Out[18]:

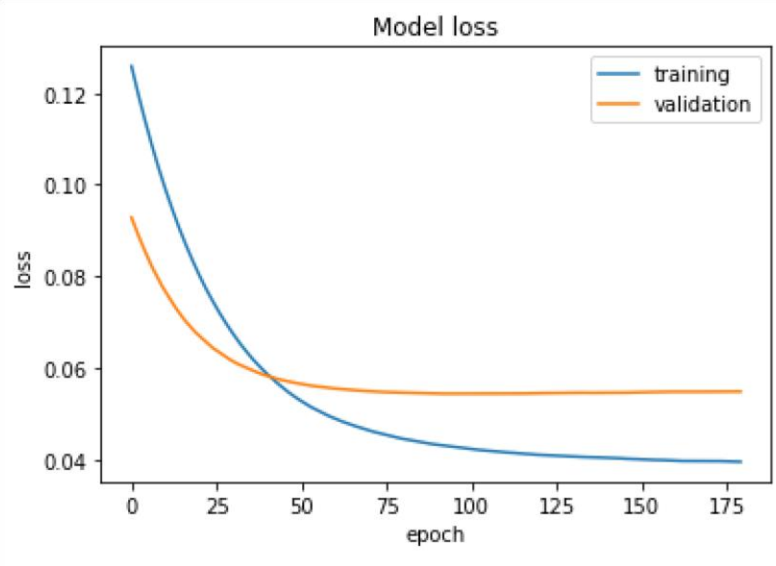


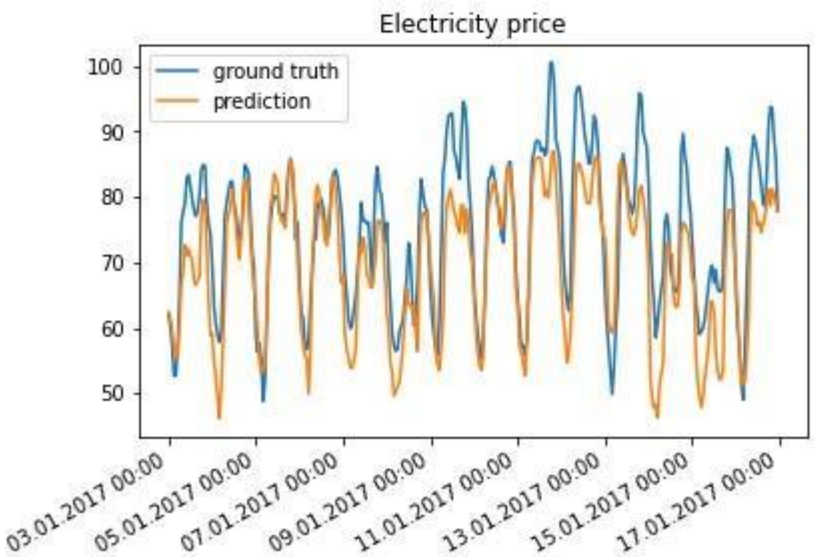


In[19]:

```
run_xgboost("price",        X_attr_time_train_norm_windows,
Y_price_train_norm,
      X_attr_time_val_norm_windows, Y_price_val_norm,
      X_attr_time_test_norm_windows,
Y_price_test_norm,
      Y_price_scaler, Y_price_val, Y_price_test)
```

Out[19]:



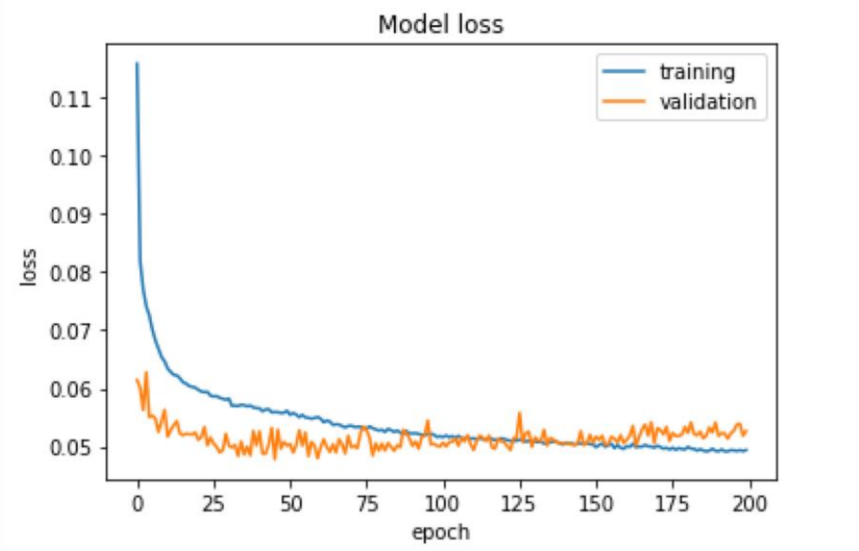


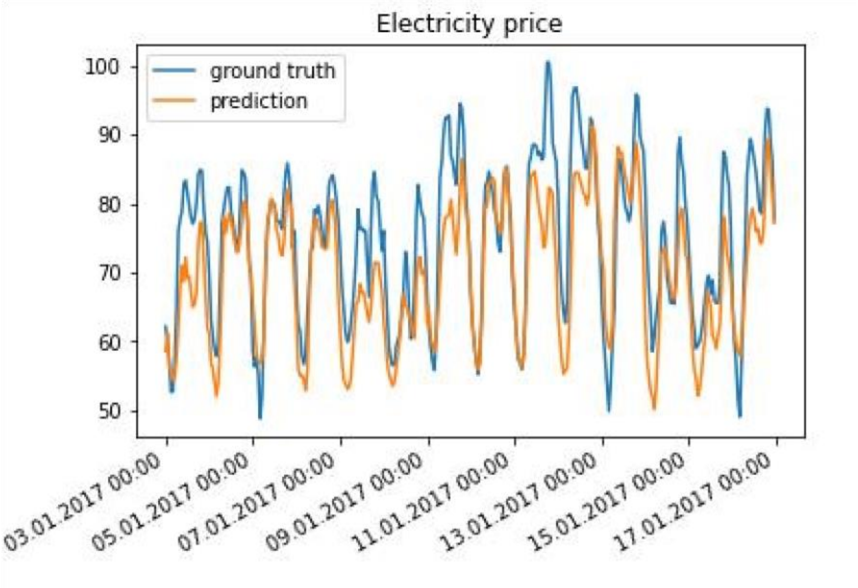
In[20]:

```
run_model(gru_concat_time_price_model, "price",
          [X_attr_train_norm_windows,
X_time_train_norm_values], Y_price_train_norm,
          [X_attr_val_norm_windows,
X_time_val_norm_values], Y_price_val_norm,
```

```
            [X_attr_test_norm_windows,
X_time_test_norm_values], Y_price_scaler,
Y_price_val, Y_price_test)
```

Out[20]:

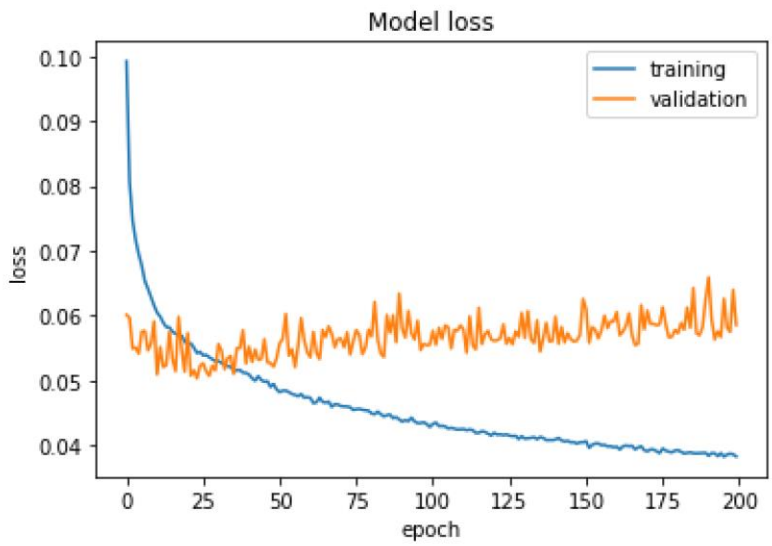


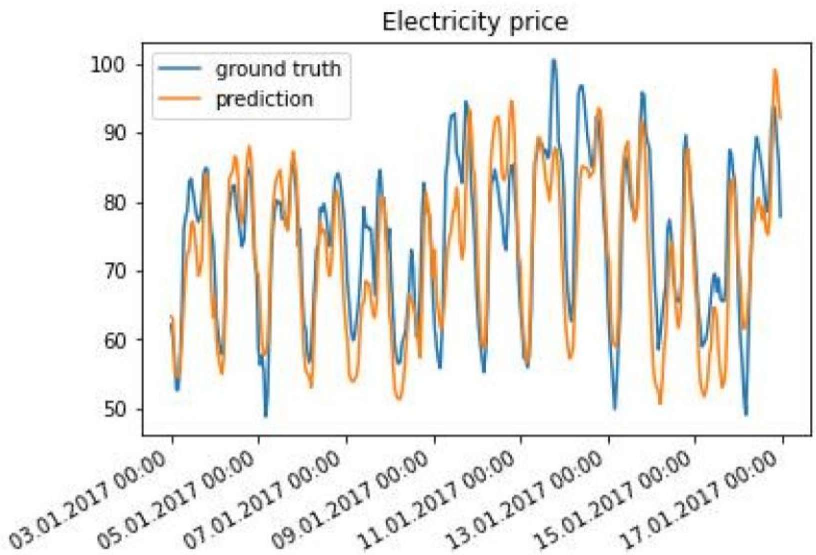


In[21]:

```
run_model(cnn_concat_time_price_model, "price",
     [X_attr_train_norm_windows,
X_time_train_norm_values], Y_price_train_norm,
     [X_attr_val_norm_windows,
X_time_val_norm_values], Y_price_val_norm,
[X_attr_test_norm_windows,
X_time_test_norm_values], Y_price_scaler, Y_price_val,
Y_price_test)
```

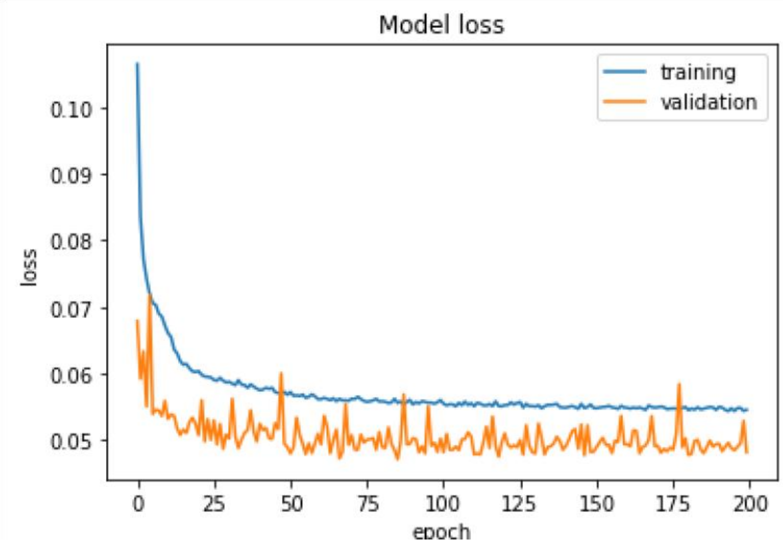
Out[21]:

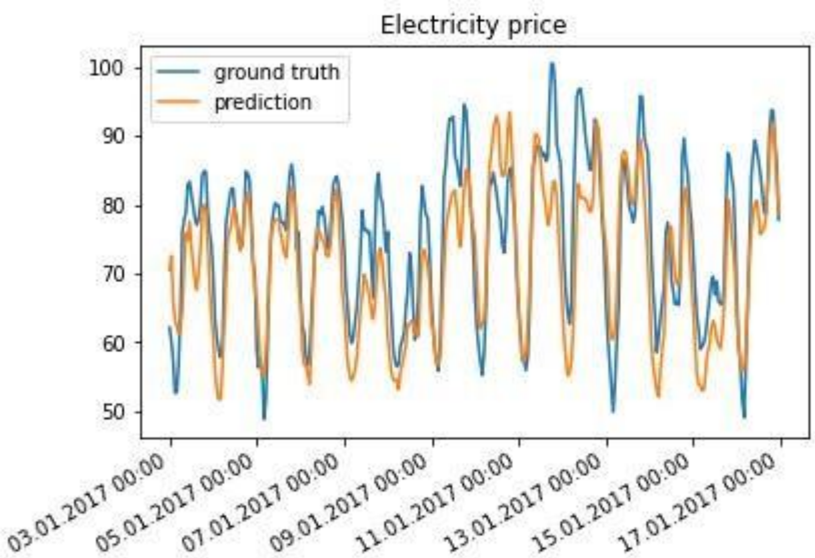




In[22]:

```
run_model(conv_concat_time_price_model, "price",
[X_attr_train_norm_windows,
X_time_train_norm_values], Y_price_train_norm,
[X_attr_val_norm_windows,
X_time_val_norm_values], Y_price_val_norm,
     [X_attr_test_norm_windows,
X_time_test_norm_values], Y_price_scaler, Y_price_val,
Y_price_test)
```





Conclusion:

In conclusion, loading and preprocessing of electricity price prediction data are foundational steps that set the stage for accurate and reliable forecasting models. These steps are essential in ensuring that the data is clean, relevant, and suitable for analysis. Through data loading, we bring the raw data into our analysis environment, making it accessible for further manipulation. Preprocessing then allows us to transform, clean, and shape the data into a format that's conducive to building effective predictive models.

The significance of these steps lies in their capacity to enhance the predictive accuracy of models, their robustness to noise and outliers, and their ability to handle the temporal nature of electricity price data. Feature engineering and dimensionality reduction, part of preprocessing, enable us to uncover valuable insights, creating a bridge between raw data and actionable predictions. Furthermore, handling missing data, outliers, and scaling features ensures that our models are more accurate, efficient, and ready for real-world deployment.

In the energy sector, where electricity price forecasting plays a pivotal role in decision-making and resource allocation, the quality of loading and preprocessing has a direct impact on market participation, sustainability, and operational efficiency. By following best practices in data loading and preprocessing, we pave the way for more informed and effective strategies in energy management, ultimately contributing to a more sustainable and reliable energy future.