Machine Learning bei Zeitreihen

Seminararbeit

für die Prüfung zum
Bachelor of Science

des Studienganges Informatik
an der Dualen Hochschule Baden-Württemberg Stuttgart

von

Maximilian Welzel

26.04.2024

Bearbeitungszeitraum	12 Wochen
Matrikelnummer, Kurs	3365934, TINF21A
Ausbildungsfirma	Festo, Esslingen

Erklärung

gemäß § 5 (4) der „Studien- und Prüfungsordnung DHBW Technik“ vom 14. Juli 2021.

Ich habe die vorliegende Projektarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

Ort	Datum	Unterschrift
Notzingen	25.04.2024	Max Welzel

Abstract

Diese Seminararbeit bietet einen umfassenden Überblick über den Einsatz von maschinellem Lernen bei der Analyse und Vorhersage von Zeitreihendaten. Im Fokus stehen dabei die spezifischen Herausforderungen und Methoden, die bei der Arbeit mit Zeitreihendaten relevant sind, einschließlich der Behandlung von fehlenden Werten und der Analyse von Trends und Saisonalität. Es werden drei verschiedene maschinelle Lernverfahren untersucht: ARIMA als Vertreter klassischer statistischer Modelle, Random Forest als Methode des überwachten Lernens und Long Short-Term Memory (LSTM) aus dem Bereich der tiefen neuronalen Netze. Jedes dieser Modelle wird anhand von Beispieldaten implementiert, um deren Eignung für die Vorhersage von Zeitreihendaten zu demonstrieren. Die Arbeit zeigt auf, wie unterschiedlich die Modelle mit den Herausforderungen der Zeitreihenanalyse umgehen und bewertet ihre Leistungsfähigkeit im Kontext eines spezifischen Anwendungsbeispiels.

Inhaltsverzeichnis

Abkürzungsverzeichnis

LSTM Long Short-Term Memory

Abbildungsverzeichnis

1 Einleitung

Das Ziel dieser Arbeit ist es, einen Einblick in die Arbeit mit Zeitreihendaten im Kontext des maschinellen Lernens zu erhalten. In einer Voruntersuchung sollen der Aufbau und die Besonderheiten der benötigten Komponenten beschrieben werden. Darüber hinaus soll ein Einblick in die verschiedenen Verfahren gewährt werden. Dieser Einblick erfolgt mit Hilfe von Beispieldaten und ausgewählten Verfahren durch eine einfache Implementierung. Die Implementierung findet in einem Jupyter Notebook statt, welches im Anhang zu finden ist.

2 Voruntersuchung

2.1 Zeitreihendaten

Zeitreihendaten sind eine spezielle Art von Daten, die über aufeinanderfolgende Zeitpunkte hinweg erfasst werden. Typischerweise besteht eine Zeitreihe aus einer Sequenz von Datenpunkten, die in gleichmäßigen Zeitabständen gemessen werden [1]. Zeitreihendaten werden in verschiedenen Bereichen, wie dem Aktienmarkt, der Wetterüberwachung durch Sensoren oder der Überwachung von Gehirnaktivität, eingesetzt [2]. Bei der Betrachtung der aufgenommenen Daten werden die Zeitpunkte der Datenaufnahme stets mit aufgezeichnet. Dabei kann erwartet werden, dass die zeitlichen Abstände der Daten immer gleich sind und diese auch zeitlich sortiert gespeichert werden. Das Aufzeichnen des Aufnahmezeitpunktes erhöht die Aussagekraft eines Datenpunktes erheblich. Werden über einen längeren Zeitraum mehrere Daten aufgenommen, ermöglichen diese die Analyse der Daten. Zeitreihendaten, die in gleichmäßigen Intervallen erfasst werden, ermöglichen es, nachträglich Analysen durchzuführen. Diese können Organisationen dabei unterstützen, Leistungen zu bewerten und zu optimieren, Risiken zu minimieren und operative Entscheidungen zu verbessern. Die genaue Zeitregistrierung jedes Datenpunkts vereinfacht es zudem, Ereignisse in Beziehung zu setzen und Muster sowie Zusammenhänge zu erkennen. Aufgrund dieser Eigenschaften können Zeitreihendaten auch genutzt werden, um zukünftige Entwicklungen und Trends vorherzusagen. [3]

2.2 Maschinelles Lernen mit Zeitreihendaten

Bei der Arbeit mit maschinellem Lernen in Verbindung mit Zeitreihendaten ist zu beachten, dass sich der Umgang mit diesen Daten von dem mit zeitunabhängigen Daten unterscheidet. Diese zeitliche Abhängigkeit muss beispielsweise bei der Behandlung fehlender Werte berücksichtigt werden. Fehlt ein Wert in der Zeitreihe, kann dieser nicht einfach durch den Durchschnitt ersetzt werden, wie es bei statischen Daten der Fall wäre, da dies zur Veränderung zukünftiger Werte führen könnte. Stattdessen sollten andere Verfahren zur Schließung dieser Lücken verwendet werden.[4] Eine Möglichkeit besteht darin, die Interpolationsmethode zu verwenden. Bei dieser Methode werden zwei bekannte Datenpunkte durch eine gerade Linie verbunden und der Wert an einem Zwischenpunkt durch einfache lineare Gleichungen geschätzt. Dadurch werden die fehlenden Werte mit Werten ersetzt, die passend zum Verlauf der Zeitreihe passen.[5] Bei den Modellen bzw. den Daten wird zwischen univarianten und multivarianten Zeitreihenmodellen unterschieden. Univariante Modelle erstellen Untersuchungen und Vorhersagen basierend auf einer einzigen Variablen. Hingegen wird bei einem multivarianten Modell mehr als eine Variable verwendet werden, um Vorhersagen zu treffen.[6] Die vorliegenden Beispieldaten sind univariant, weshalb sich die Implementierungen auf diese beschränken.

2.3 Komponenten von Zeitreihendaten

Bei Zeitreihendaten, betrachtet im Hinblick auf Vorhersagen, sind die Komponenten Trend und Saisonalität von hoher Bedeutung. Der Trend einer Zeitreihe gibt die Richtung der Datenveränderung über die Zeit an. Bei der Saisonalität, geht es um periodisch auftretende Verhaltensweisen. Mögliche Beispiele dafür sind soziale/kulturelle Ereignisse wie Ferien oder Naturereignisse wie Wetterschwankungen. Können die Daten nicht zu Trend oder Saisonalität zugeordnet werden werden diese als Restwerte oder Resid bezeichnet. In Abbildung 2.1 kann eine Unterteilung eines Datensatzes in die drei verschiedenen Komponenten betrachtet werden. Bei dem Datensatz der Abbildung liegen Trend sowie Saisonalität vor. [7]

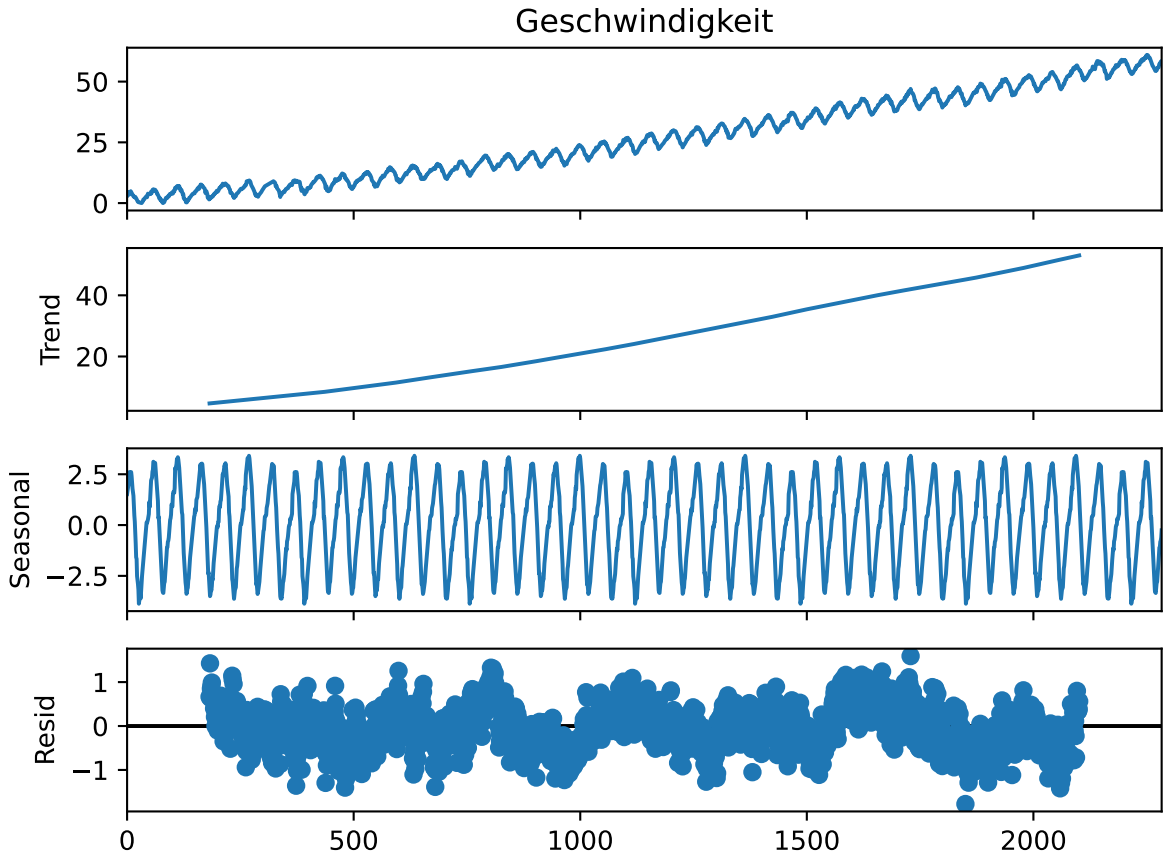


Abbildung 2.1: Komponenten von Zeitreihen

Zeitreihendaten werden als stationär bezeichnet, wenn keine Veränderungen in den statistischen Merkmalen wie Durchschnitt und Varianz vorzuweisen sind [7]. Durch das Gleichbleiben des Durchschnitts können sich die Daten nicht in eine Richtung verändern. Somit kann kein Trend vorliegen, wenn davon gesprochen wird, dass die Zeitreihendaten stationär sind.[8]

3 Implementierung

3.1 Wahl der maschinellen Lernverfahren

Bei der Recherche zu verschiedenen Verfahren sind viele Herangehensweisen aufgefallen. Um mit dieser Arbeit einen Überblick über die Methoden zu verschaffen, lassen sich diese bzw. die verschiedenen zur Verfügung stehenden Modelle in drei Kategorien unterteilen. Die Kategorien werden auf Basis von [9] und [8] gewählt. [9] bietet einen Überblick über relative Leistung bei der Zeitreihenprognose und unterteilt dabei die Herangehensweisen in klassische statistische Methoden, Machine Learning- und Deep Learning-Techniken. Auch [8] unterteilt die Modelle in klassische Zeitreihen-, überwachtes Machine Learning- und Deep Learning Modelle. Auf dieser Grundlage soll in dieser Arbeit ein Überblick über die verschiedenen Verfahren gegeben werden, indem für jede Kategorie mit Beispieldaten eine Prognose durchgeführt wird. Die Wahl der präzisen Modelle fällt bei den klassischen, statistischen Modellen auf ARIMA. ARIMA wird gewählt, da es bei der Recherche vermehrt aufgetreten ist und in einigen Quellen als beliebete Altbekannte Methode bezeichnet wird(vgl. [1],[9],[10]). Bei den Deep Learning-Verfahren wird in vielen Quellen Long Short-Term Memory (LSTM) Methode als Beispeil dafür verwendet. Deshalb viel hierbei die Wahl auf dieses Verfahren (vgl. [9],[10],[8]). Bei den überwachten Machine Learning-Verfahren fällt die Wahl auf Random-Forest, da es sich dabei um ein bekanntes Verfahren handelt [8].

3.2 Datenvorbereitung

In der tatsächlichen Implementierung beispielhafter maschineller Lernverfahren sollen Messwerte mit folgendem Format betrachtetet werden:

```
{
    "fin": "F1000001",
    "zeit": 169839374,
    "geschwindigkeit": 33,
    "ort": "Stuttgart"
}
```

Listing 1: Datenformat der Beispieldaten

Da die vorliegenden Beispieldaten in zu geringer Anzahl vorhanden sind und zufällige Werte für die Geschwindigkeit genutzt werden, können diese für die Vorstellung der Verfahren nicht verwendet werden. Für die Vorstellung der verschiedenen Verfahren müssen passende Werte vorliegen, bei denen auch ein gutes Ergebnis erzielt werden kann. Die Beispieldaten bestehen aus CO2-Messwerten (siehe [11]), die auf ein Fahrzeugszenario angepasst werden. Dafür wurde das vorliegende Zeitformat der CO2-Messwerte in das Unix-Timestamps-Format umgewandelt und die Geschwindigkeit variiert zwischen 0 und 60 km/h . Es handelt sich letztlich um etwas

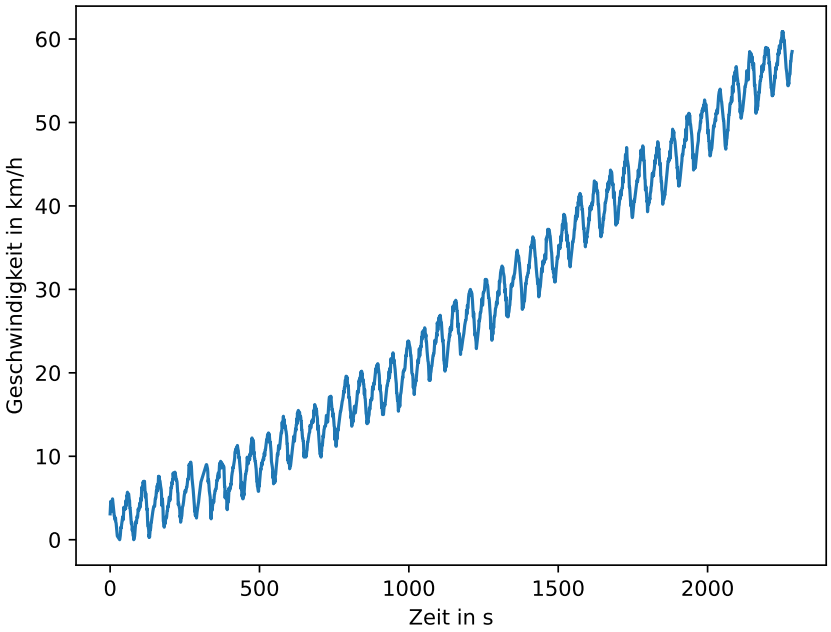


Abbildung 3.1: Verlauf der Beispieldaten

mehr als 2000 Datensätze, die für die weitere Untersuchung verwendet werden. In der Untersuchung wird nur der Verlauf der Geschwindigkeit eines Fahrzeugs betrachtet, weshalb die weiteren Informationen im vorliegenden Format nicht weiter in der Untersuchung betrachtet werden. Nachdem die Daten in das gewünschte Format gebracht werden, wird die tatsächliche Datenvorbereitung durchgeführt. Als erster Schritt wird nach fehlenden Werten innerhalb der Daten gesucht. Da fehlende Werte vorliegen, müssen diese beseitigt werden. In Kapitel 2.2 wird erläutert, dass diese fehlenden Werte nicht einfach durch beispielsweise einen allgemeinen Durchschnitt ersetzt werden können. Deswegen wurde hierfür die Interpolationsfunktion verwendet (siehe Listing 2), welche auch als Beispiel in Kapitel 2.2 beschrieben wird. Nach dem Auffüllen der fehlenden Werte kann der lückenlose Verlauf der Beispieldaten in Abbildung 3.1 betrachtet werden.

```
1 car_data = car_data.fillna(car_data.interpolate())
```

Listing 2: Aufruf der interpolate-Funktion

3.3 Datenanalyse

In der Datenanalyse werden nun die Daten genauer betrachtet, insbesondere im Bezug auf Trend, Saisonalität und Residuen. In Abbildung 3.2 kann eine Dekomposition der einzelnen Komponenten betrachtet werden. Es wird schnell ersichtlich, dass ein aufsteigender Trend vorliegt. In den Daten lässt sich zudem auch ein klares Muster in der Saisonalität erkennen. Jedoch

gibt es eine Vielzahl an Datenpunkten, die nicht direkt einer der Kategorien zugeordnet werden können. Ein wichtiger Teil der Datenanalyse ist es ebenfalls zu prüfen, ob die Datenreihe stationär ist. Da ein klarer Trend in der vorherigen Untersuchung zu erkennen ist, kann davon ausgegangen werden, dass es sich bei den Daten nicht um stationäre Daten handelt. Im Code im Anhang wurde jedoch ein Test durchgeführt, der dies ebenfalls bestätigt, worauf hier nicht weiter eingegangen wird.

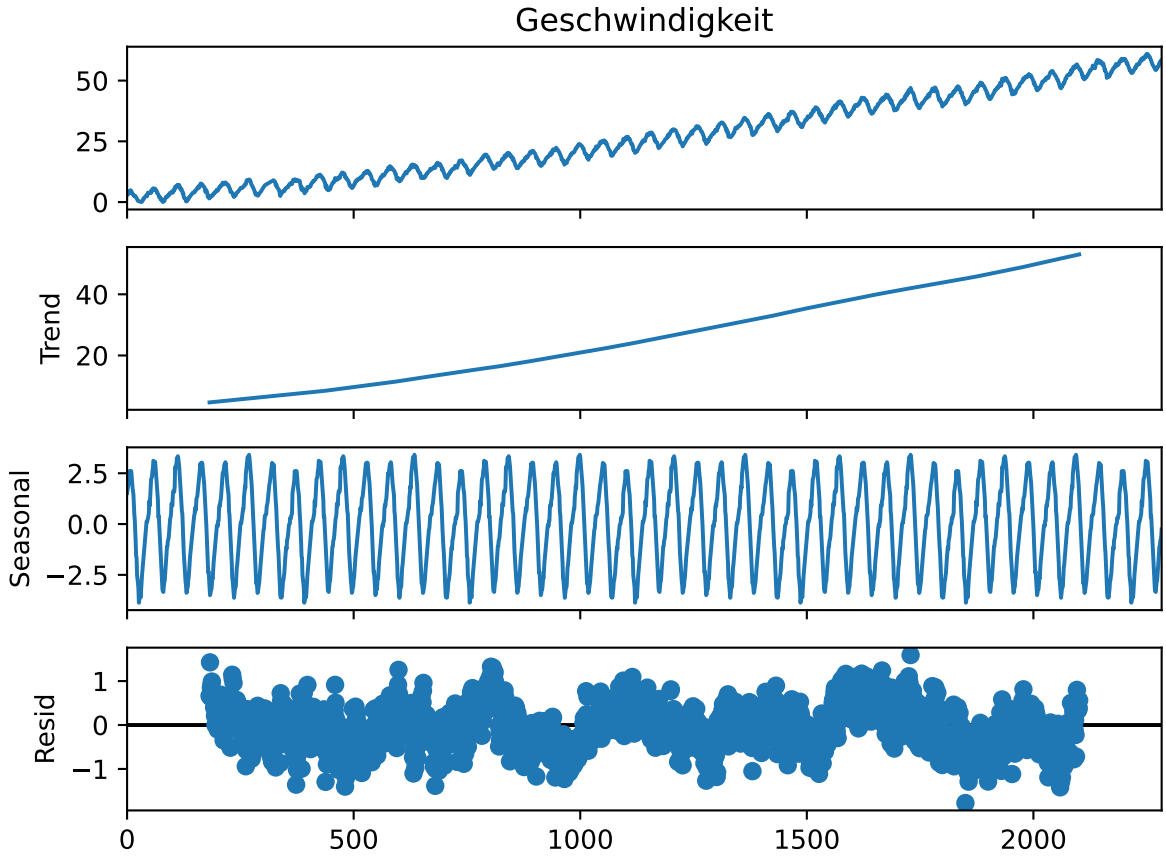


Abbildung 3.2: Dekomposition der Komponenten

3.4 Modellierung

Für den nächsten Schritt werden beispielhaft drei verschiedene Verfahren zur Erstellung von Modellen mittels maschineller Lernverfahren betrachtet. Dabei konzentriert sich die Analyse auf die Auswertung der Vorhersagen der Modelle. Der zugehörige Code mit Beschreibungen kann im Anhang eingesehen werden, da eine detaillierte Darstellung den Rahmen dieser Seminararbeit sprengen würde.

3.4.1 ARIMA

ARIMA wurde als Vertreter der klassischen statistischen Modelle gewählt. Hier ist zu Beginn zu erwähnen, dass die Datenpunkte für das Erstellen dieses Modells reduziert werden müssen, da der Computer, auf dem die Modellierung durchgeführt wird, nicht genügend Ressourcen für diese Aufgabe hat. Bei diesem Modell werden letztlich nur 600 Daten betrachtet. In Abbildung 3.3 kann das Ergebnis des Modells betrachtet werden. Die Trainingsdaten, mit denen das Modell trainiert wird, sind in blau dargestellt. In grün sind die Testdaten zu sehen, beziehungsweise die tatsächlichen Daten der Testphase. In rot sind die vom Modell vorhergesagten Testdaten zu sehen. Betrachtet man die vorhergesagten Werte, veralufen diese ansteigend, wie es der Trend der Daten vorgibt, jedoch weichen die Werte sich noch stark von den tatsächlichen Werten ab.

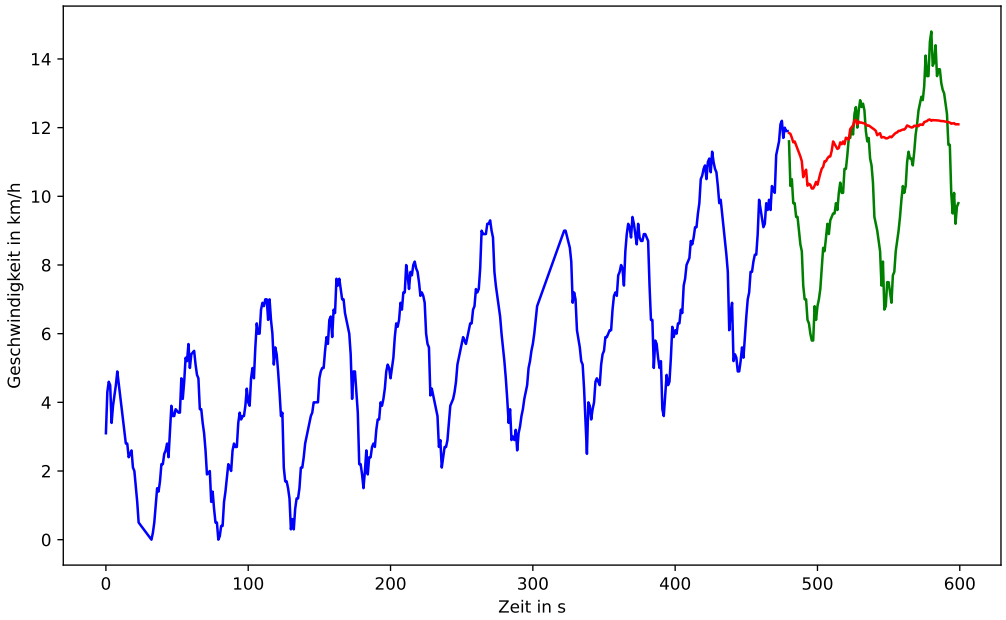


Abbildung 3.3: Visualliserung der Training-, Test- und vorhergesagten Daten mit ARIMA

3.4.2 Random-Forest

Für die klassischen maschinellen Lernverfahren wird das Random-Forest-Verfahren verwendet. Da dieses Verfahren nicht direkt mit Zeitreihendaten arbeitet und den zeitlichen Aspekt bei der Erstellung des Modells nicht berücksichtigt, müssen die zeitlichen Informationen über Variablen eingeführt werden. In diesem Fall wird der Zeitpunkt auf verschiedene Weise angegeben. In Abbildung 3.4 kann das letztendliche Ergebnis der Vorhersage betrachtet werden. Die Trainingsdaten sind in blau dargestellt, die Testdaten, mit denen nicht trainiert wurde, in grün. Die vom Modell vorhergesagten Daten sind rot gepunktet zu sehen. Es ist zu erkennen, dass

der Random-Forest sehr gut bei der Vorhersage der Werte ist, die als Trainingsdaten vorlagen. Sobald es jedoch der Wertebereich Trainingsdaten verlassen wird, bleiben die Vorhersagen auf einem Wert. Dies könnte darauf hindeuten, dass es sich hierbei um Overfitting handeln könnte.

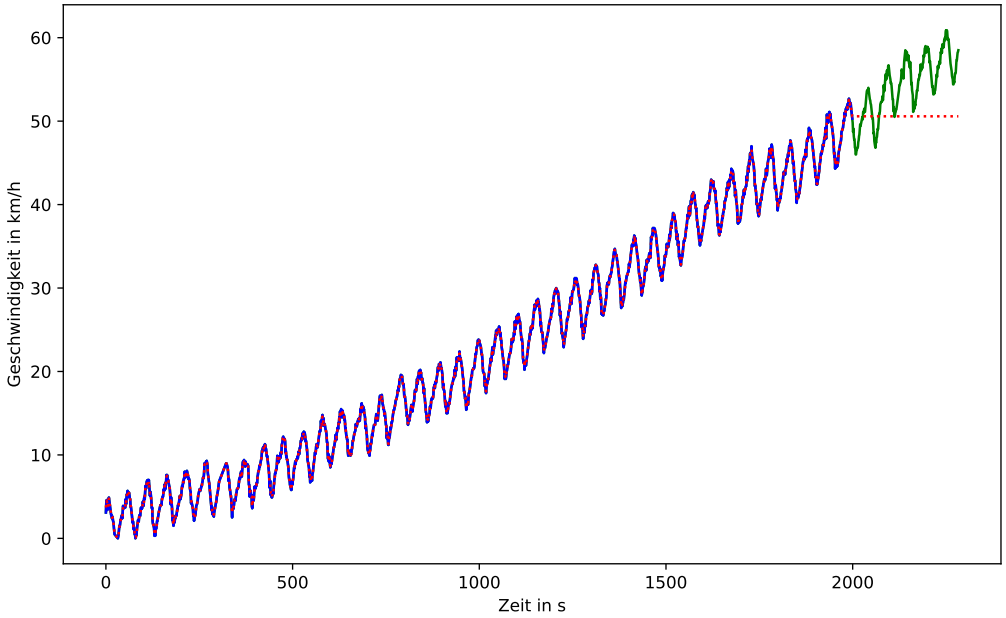


Abbildung 3.4: Visualliserung der Training-, Test- und vorhergesagten Daten mit Random-Forest

3.4.3 Long Short-Term Memory LSTM

Als letzte Methode wird das LSMT-Verfahren verwendet. Dieses Verfahren ist ein neuronales Netz, mit dem Zeitreihendaten vorhergesagt werden können. In Abbildung 3.5 sind die tatsächlichen Werte in blau zu sehen. Die Vorhersagen, die auf Basis der Trainingsdaten gemacht werden, sind in grün dargestellt und liegen ziemlich genau über dem blauen Graphen, weshalb man hier fast nur den grünen sieht. Die vorhergesagten Daten, die nicht zum Trainieren verwendet werden, sind rot gepunktet dargestellt. In beiden Fällen ist ein sehr gutes Ergebnis zu beobachten. Bei der roten Linie ist zu erkennen, dass sie umso ungenauer wird, je weiter sie sich von den Trainingsdaten entfernt. Dennoch ist weiterhin ein ähnlicher Trend sowie ein Saisonalität im Verlauf des Graphen zu erkennen.

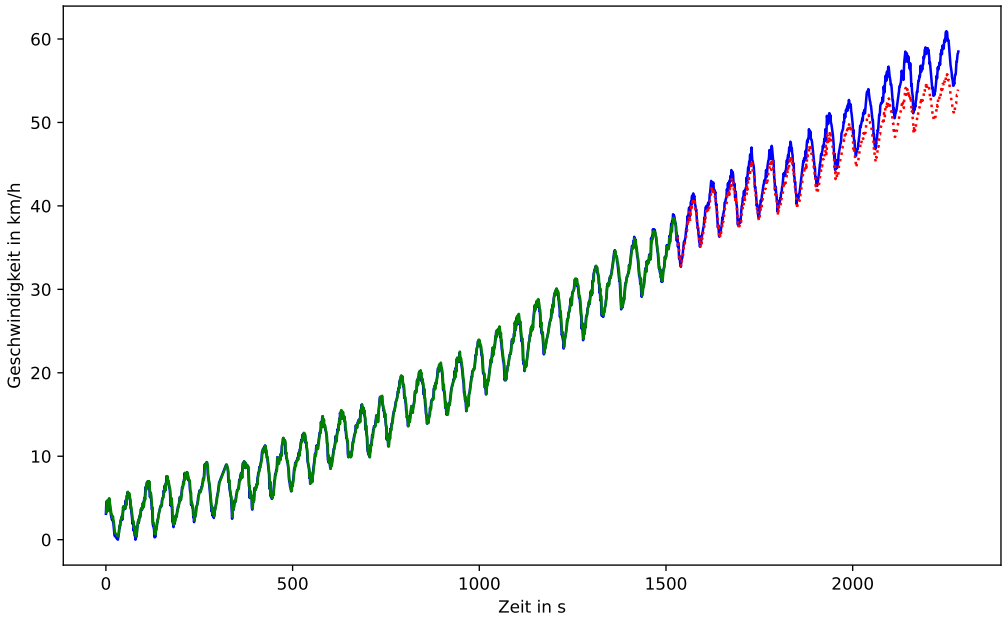


Abbildung 3.5: Visualliserung der Training-, Test- und vorhergesagen Daten mit LSTM

4 Fazit

Zusammenfassend wurde in dieser Arbeit ein Einblick über die verschiedenen Methoden zum Arbeiten mit maschinellen Lernverfahren in Bezug auf Zeitreihendaten vorgestellt und deren Unterschied zu der Arbeit mit klassischen maschinellen Lernverfahren aufgezeigt. In der Literatur gibt es viele unterschiedliche Verfahren, die verwendet werden können und letztlich immer an den Verwendungszweck angepasst werden müssen. Von den drei näher betrachteten Methoden hat die Random-Forest-Methode am schlechtesten abgeschnitten. Dabei ist jedoch zu beachten, dass diese die einzige Methode war, die nicht für Zeitreihendaten vorgesehen war. Bei dem ARIMA-Verfahren hat die Vorhersage einigermaßen gut funktioniert, jedoch konnten nicht alle Daten verwendet werden, da die Methode zu viel Leistung erfordert hat. Um alle vorliegenden Daten für die Vorhersagen zu verwenden, wird mehr Leistung gebraucht, was dadurch auch als Nachteil im Vergleich mit den anderen Methoden betrachtet wird. Abschließend kann die LSTM-Methode in dieser Vorstellung der Methoden als Favorit angesehen werden. Die Methode konnte alle Daten für die Erstellung des Modells nutzen und hatte im Vergleich die besten Vorhersagen. Jedoch ist zu beachten, dass bei allen Methoden noch unterschiedliche Anpassungen durchgeführt werden können, um die Vorhersagen zu verbessern. Diese Arbeit soll nicht als Empfehlung für die Wahl einer Methode genutzt werden, da für die beste Nutzung der Methode ein tiefgründigere Untersuchung der einzelnen Eigenschaften zu empfehlen ist.

5 Literaturverzeichnis

[1] Claudio Hartmann u. a. „CSAR: The Cross-Sectional Autoregression Model“. In: *2017 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*. Okt. 2017, S. 232–241. DOI: 10.1109/DSAA.2017.27. URL: https://ieeexplore.ieee.org/document/8259782 (besucht am 14. 04. 2024).

[2] *Machine Learning for Time Series Data in Python Course*. en-us. URL: https://www.datacamp.com/courses/machine-learning-for-time-series-data-in-python (besucht am 14. 04. 2024).

[3] Ted Dunning und Ellen Firedman. *Time Series Databases - New Ways to Store and Access Data*. Sebastopol: O´Reilly Media, 2015. ISBN: 978-1-4919-1702-2.

[4] Aayush Bajaj. *Time Series Prediction: How Is It Different From Other Machine Learning? [ML Engineer Explains]*. en-US. Juli 2022. URL: https://neptune.ai/blog/time-series-prediction-vs-machine-learning (besucht am 14. 04. 2024).

[5] *Interpolation: Definition, Mathe & Methoden*. de-DE. URL: https://www.studysmarter.de/schule/mathe/analysis/interpolation/ (besucht am 19. 04. 2024).

[6] Aishwarya Singh. *Multivariate Time Series Analysis With Python for Forecasting and Modeling (Updated 2024)*. en. Sep. 2018. URL: https://www.analyticsvidhya.com/blog/2018/09/multivariate-time-series-guide-forecasting-modeling-python-codes/ (besucht am 14. 04. 2024).

[7] Akshay P. Jain. *Time Series Forecasting: Data, Analysis, and Practice*. en-US. Juli 2022. URL: https://neptune.ai/blog/time-series-forecasting (besucht am 16. 04. 2024).

[8] Joos Korstanje. *How to Select a Model For Your Time Series Prediction Task [Guide]*. en-US. Sep. 2022. URL: https://neptune.ai/blog/select-model-for-time-series-prediction-task (besucht am 16. 04. 2024).

[9] Vaia I. Kontopoulou u. a. „A Review of ARIMA vs. Machine Learning Approaches for Time Series Forecasting in Data Driven Networks“. en. In: *Future Internet* 15.8 (Aug. 2023). Number: 8 Publisher: Multidisciplinary Digital Publishing Institute, S. 255. ISSN: 1999-5903. DOI: 10.3390/fi15080255. URL: https://www.mdpi.com/1999-5903/15/8/255 (besucht am 16. 04. 2024).

[10] Yennhi95zz. *A Guide to Time Series Models in Machine Learning: Usage, Pros, and Cons*. en. Aug. 2023. URL: https://medium.com/@yennhi95zz/a-guide-to-time-series-models-in-machine-learning-usage-pros-and-cons-ac590a75e8b3 (besucht am 14. 04. 2024).

[11] Josef Perktold u. a. *statsmodels/statsmodels: Release 0.14.2*. Apr. 2024. DOI: 10.5281/ZENODO.593847. URL: https://zenodo.org/doi/10.5281/zenodo.593847 (besucht am 19. 04. 2024).

6 Anhang

Daten Erstellung

In [ ]:
```
import pandas as pd
import statsmodels.datasets.co2 as co2
import matplotlib.pyplot as plt
```

In [ ]:
```
co2_data = co2.load().data
print(co2_data)

print(co2_data['co2'].max())
print(co2_data['co2'].min())
print(co2_data['co2'].max() - co2_data['co2'].min())
```

```
              co2
1958-03-29  316.1
1958-04-05  317.3
1958-04-12  317.6
1958-04-19  317.5
1958-04-26  316.4
...           ...
2001-12-01  370.3
2001-12-08  370.8
2001-12-15  371.2
2001-12-22  371.3
2001-12-29  371.5

[2284 rows x 1 columns]
373.9
313.0
60.89999999999998
```

Um Daten zu haben, welche repräsentativ für die Messdaten stehen, werden Daten heruntergeladen, die auf mögliche Messdaten für ein Fahrzeug angepasst werden. Da die Seminararbeit und der dazugehörige Code für eine Vorstellung von Vorgehen genutzt wird, werden Daten genommen, bei denen bekannt ist, dass sie betimmte Muster vorweisen.

In [ ]:
```
co2_data['co2_adjusted'] = co2_data['co2'] - co2_data['co2'].min()
car_data = co2_data['co2_adjusted']

car_data = car_data.rename('Geschwindigkeit')


car_data.index = pd.date_range(start='2022-01-01', periods=len(car_data), freq='1s'
car_data_random_forest = car_data.copy()
car_data_random_forest.index  = pd.date_range(start='2022-01-01', periods=len(car_d

car_data.index =  (car_data.index.astype('int64') // 10**9).astype('int32')

print(car_data)
```

```
1640995200     3.1
1640995201     4.3
1640995202     4.6
1640995203     4.5
1640995204     3.4
               ...
1640997479    57.3
1640997480    57.8
1640997481    58.2
1640997482    58.3
1640997483    58.5
Name: Geschwindigkeit, Length: 2284, dtype: float64
```

Bei der Anpassungen der Daten, wurde nur die Differnez der vorherigen Daten verwendet, damit Werte gewonnen werden, welche eher zu der Geschwindigkeit eines Autos passt. Damit die zeitlichen Abstände zum Beispiel passen,wird der Abstand der Einträge auf 1 Sekunde gesetzt. Für die weiteren Schritte wird angenommen, dass aus der Tabelle "Messung"(aus der Vorlesung) die Messdaten eines Fahrzeugs entnommen werden und weiter untersucht werden. Der Timestep werden dafür auch in das Unix Timestamp-Format umgewandelt.

Maschinelles Lernen mit Zeitreihen

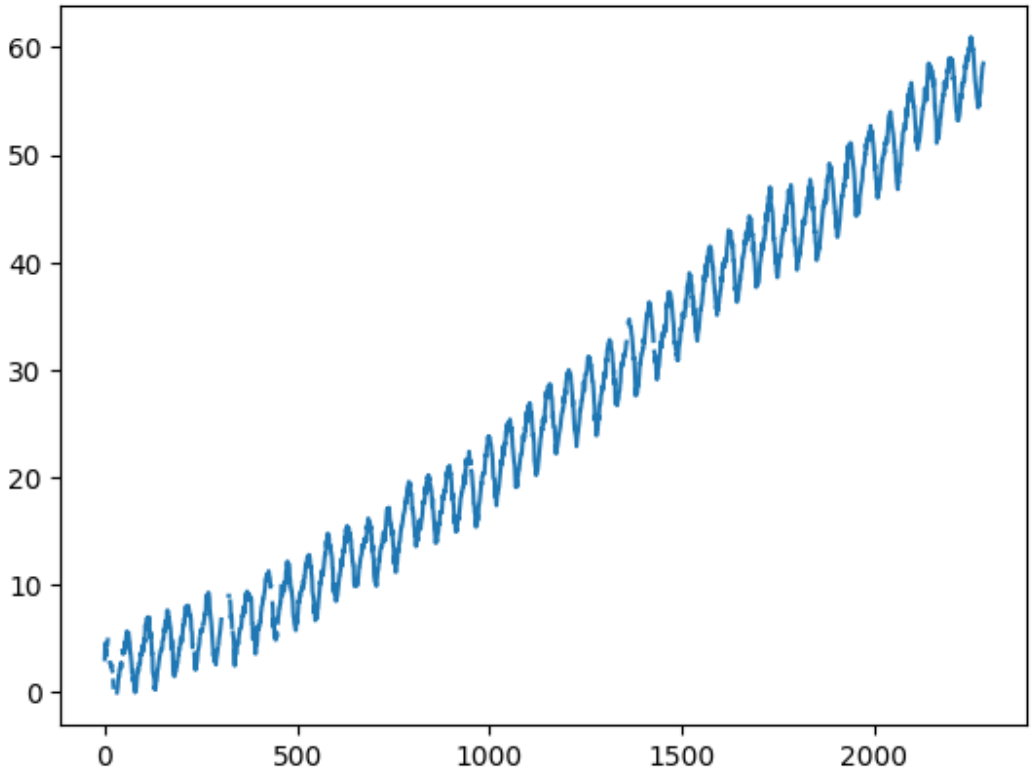
Datenvorbereitung

In [ ]:
```
car_data.index = car_data.index - car_data.index.min()
car_data
```

Out[ ]:
```
0        3.1
1        4.3
2        4.6
3        4.5
4        3.4
        ...
2279    57.3
2280    57.8
2281    58.2
2282    58.3
2283    58.5
Name: Geschwindigkeit, Length: 2284, dtype: float64
```

Um ein besseres zeitliches Verständis zu erhalten, wird von der Zeitangabe der min-Wert abgezogen. Dadurch entspricht die Zeit die Vergangen Sekunden seit dem Start der Aufnahme der Daten.

In [ ]:
```
car_data.plot()
plt.savefig('pictures/Datenvorbereitung/car_data_NaN.pdf')
plt.show()
```



Werden die Daten in einem Graphen dargestellt, ist sichtbar, dass Lücken in den Daten vorliegen, welche mit Werten ersetzt werden müssen.

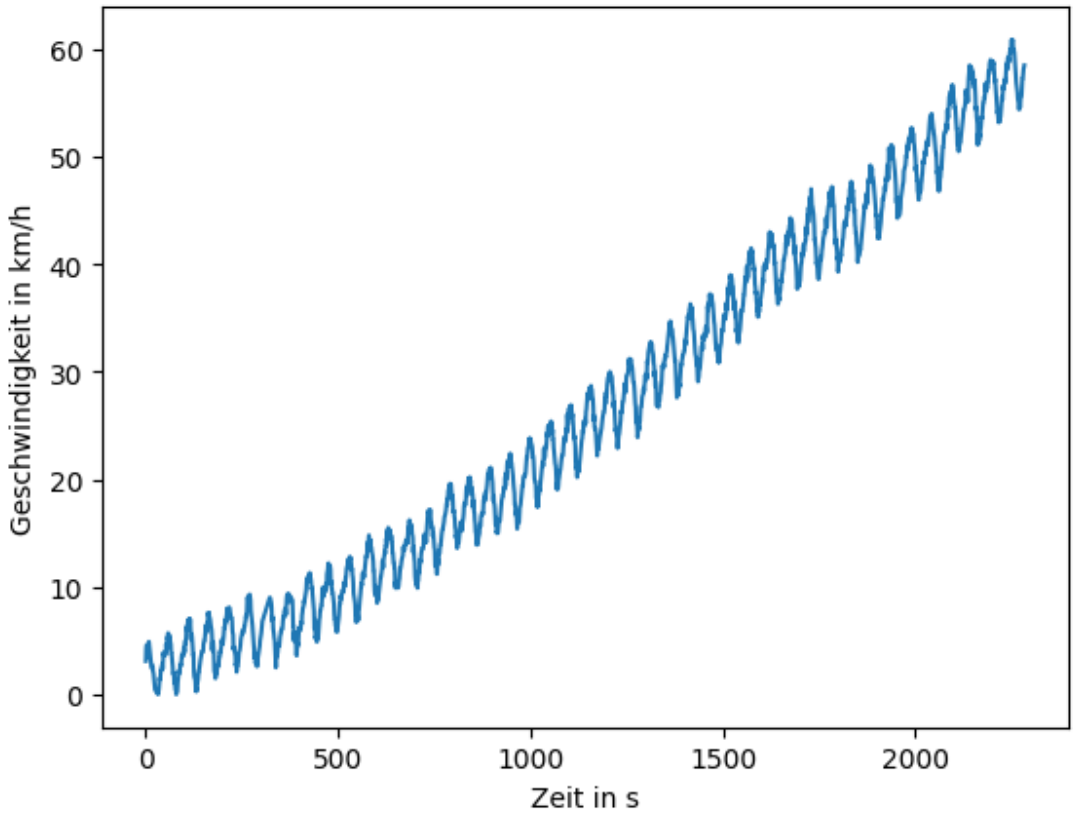
In [ ]:
```
# anzahl der Nan Werte
print("Es liegen folgende Anzahl NaN-Werte vor:")
print(car_data.isna().sum())
```

```
Es liegen folgende Anzahl NaN-Werte vor:
59
```

In [ ]:
```
car_data = car_data.fillna(car_data.interpolate())
car_data_random_forest = car_data_random_forest.fillna(car_data_random_forest.inter
```

Wie in der Seminararbeit beschrieben, können diese Werte nicht mit Durchschnittswerten ersetzt werden. Aus diesem Grund wird die Funktion interpolate() benutzt um die NaN-Werte aufzufüllen. Dabei werden zwei bekannte Datenpunkte mit einer geraden Linie verbunden und der Wert an einem Zwischenpunkt durch eine lineare Gleichungen geschätzt.

In [ ]:
```
car_data.plot()
plt.ylabel('Geschwindigkeit in km/h')
plt.xlabel('Zeit in s')
plt.savefig('pictures/Datenvorbereitung/car_data_ohneNaN.pdf')
plt.show()
```

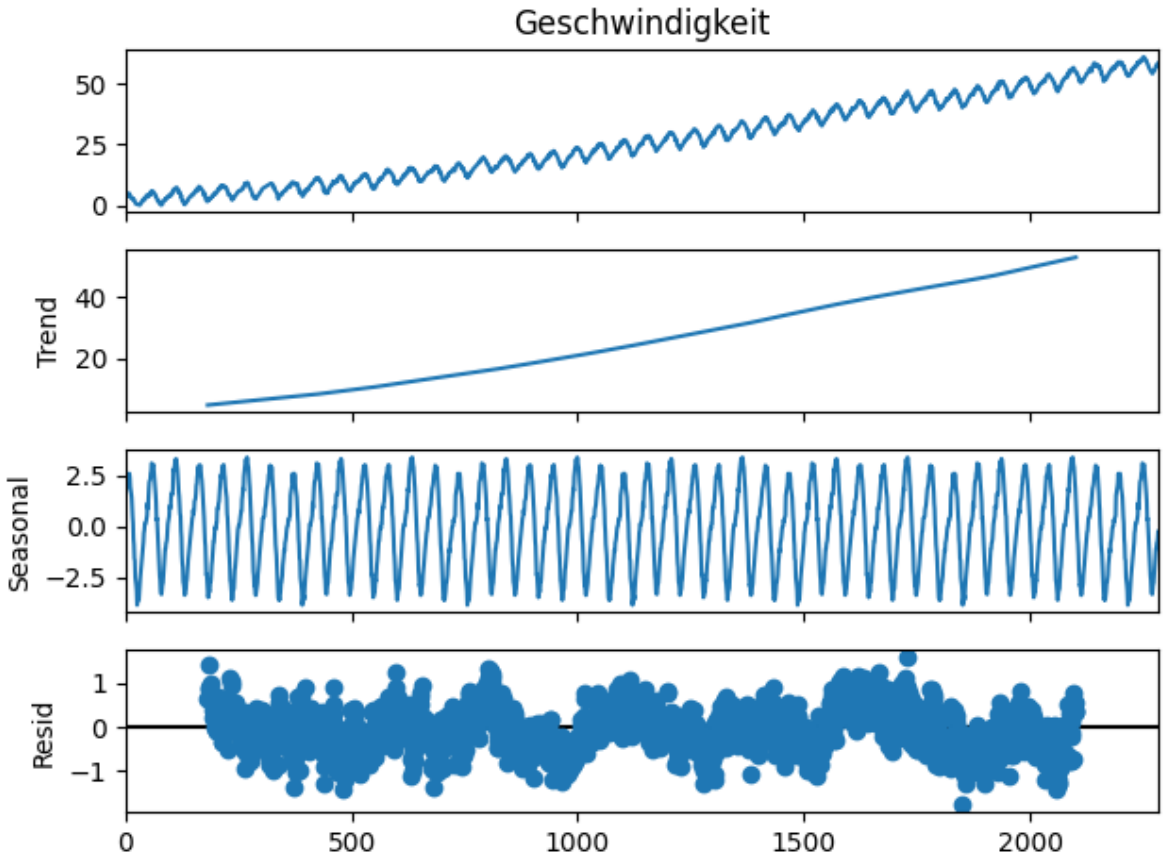


In [ ]:
```
# anzahl der Nan Werte
print("Es liegen folgende Anzahl NaN-Werte vor:")
print(car_data.isna().sum())
```

```
Es liegen folgende Anzahl NaN-Werte vor:
0
```

Datenanalyse

In [ ]:
```
from statsmodels.tsa.seasonal import seasonal_decompose

result = seasonal_decompose(car_data, model='additive', period=365)
result.plot()
# export the plot as pdf
plt.savefig('pictures/Datenanalyse/seasonal_decompose.pdf')
plt.show()
```



Im Graph sind die bereits in der Seminarabeit angesprochenen drei Komponenten der Zeitreihen zu sehen. Trend, Saisonalität und Residual.

Es ist zu sehen, dass sowohl ein Trend als auch eine Seasonality in den Daten vorliegt. Es liegen zudem aber auch Residual Werte vor, welche nicht zu einem der Kategorien zugeordndet werden konnten.

In [ ]:
```
from statsmodels.tsa.stattools import adfuller
adf, pval, usedlag, nobs, crit_vals, icbest =  adfuller(car_data.values)
print('ADF test statistic:', adf)
print('ADF p-values:', pval)
print('ADF number of lags used:', usedlag)
print('ADF number of observations:', nobs)
print('ADF critical values:', crit_vals)
print('ADF best information criterion:', icbest)
```

```
ADF test statistic: 0.03378459745816527
ADF p-values: 0.961238452828603
ADF number of lags used: 27
ADF number of observations: 2256
ADF critical values: {'1%': -3.4332519309441296, '5%': -2.8628219967376647, '10%':
-2.567452466810334}
ADF best information criterion: 2578.39090925253
```

Mit dem diesem Test kann nachgeprüft werden, ob die Zeitreihendaten stationär sind. Der Wert für test und p-Wert sind beides Werte die auf keine stationäre Daten hinweisen. Das Ergebnis ist auch zu erwarten, da in dem vorherigen Graphen bereits ein Trend zu erkennen

war.

Modellierung

ARIMA

In [ ]:
```
car_data_ARIMA = car_data[:600]
```

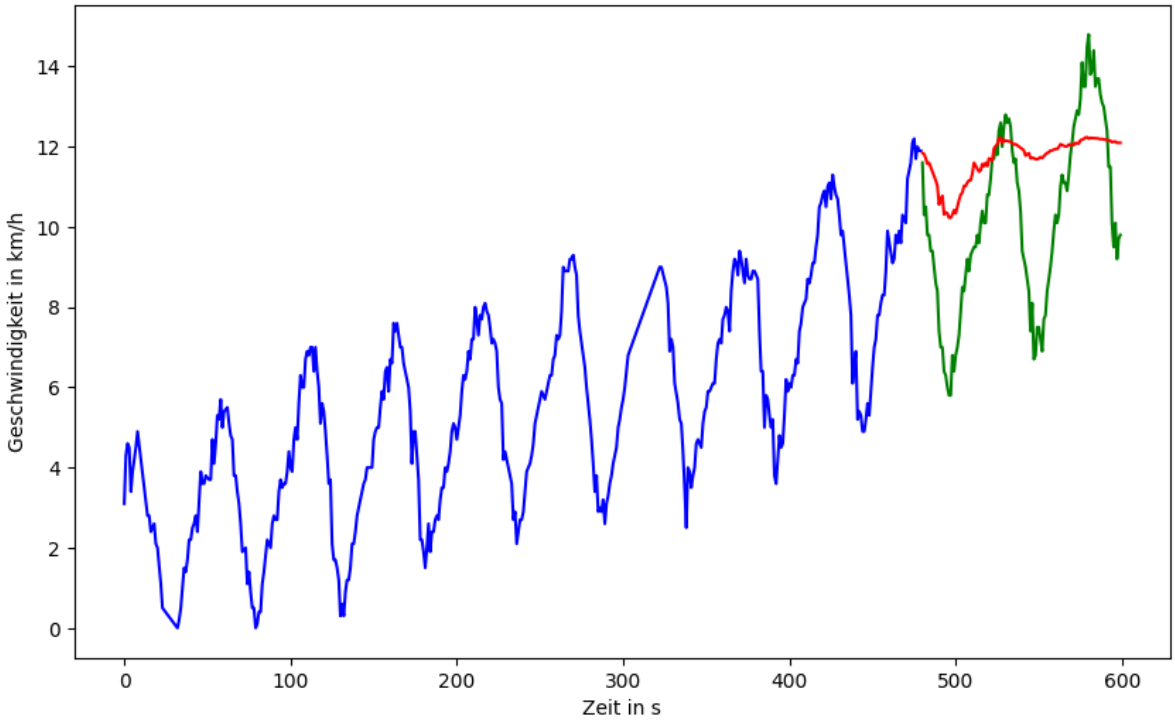
Die Daten für ARIMA mussten auf 600 verringert wurden, da die Daten sonst zu viele waren und nicht von meinem Rechner verarbeitet werden konnten.

In [ ]:
```
import pmdarima as pm
from pmdarima.model_selection import train_test_split
import numpy as np
import matplotlib.pyplot as plt


train, test = train_test_split(car_data_ARIMA, train_size=0.8)
print(train.shape)
print(test.shape)

model = pm.auto_arima(train, seasonal=True, m=52)
```

```
(480,)
(120,)
```

Die Daten werden in Trainings- und Test-Daten unterteilt und das Model mit den Daten trainiert.

In [ ]:
```
forecast = model.predict(test.shape[0])
plt.figure(figsize=(10, 6))
plt.plot(np.arange(train.shape[0]), train, color='blue')
plt.plot(np.arange(train.shape[0], train.shape[0] + test.shape[0]), test, color='gr
plt.plot(np.arange(train.shape[0], train.shape[0] + test.shape[0]), forecast, color
plt.ylabel('Geschwindigkeit in km/h')
plt.xlabel('Zeit in s')
plt.savefig('pictures/Modellierung/ARIMA.pdf')
plt.show()
```



Das Modell wird nun genutzt um die Test-Daten vorherzusagen. In blau sind hierbei die Werte zu sehen, welche für das Training verwendet wurden. In grün sind die tatsächlichen Werte, welche vorhergesagt werden sollten. In rot sind die vorhergesagten Werte zu sehen.

Betrachtet man die Werte ist auf jeden Fall zu sehen, dass die Werte wachsen wie es vom Trend zu erwarten ist, jedoch weichen sie dennoch stark von den tatsächlichen Werten ab.

Random-Forest

In [ ]:
```
import numpy as np


seconds = [x.total_seconds() for x in car_data_random_forest.index - car_data_rando
ten_seconds = [x.total_seconds() / 10 for x in car_data_random_forest.index - car_d
minutes = [x.total_seconds() / 60 for x in car_data_random_forest.index - car_data_

X = np.array([seconds, ten_seconds, minutes]).T
len(X)
```

Out[ ]:
```
2284
```

In [ ]:
```
car_data_random_forest_train = car_data_random_forest[:2000]
len(car_data_random_forest_train)
```

Out[ ]:
```
2000
```

2000 Werte werden für das Training verwendet

In [ ]:
```
import numpy as np
```

```
seconds = [x.total_seconds() for x in car_data_random_forest_train.index - car_data
ten_seconds = [x.total_seconds() / 10 for x in car_data_random_forest_train.index -
minutes = [x.total_seconds() / 60 for x in car_data_random_forest_train.index - car

X_train = np.array([seconds, ten_seconds, minutes]).T
print(len(X_train))


seconds = [x.total_seconds() for x in car_data_random_forest.index - car_data_rando
ten_seconds = [x.total_seconds() / 10 for x in car_data_random_forest.index - car_d
minutes = [x.total_seconds() / 60 for x in car_data_random_forest.index - car_data_

X_pred = np.array([seconds, ten_seconds, minutes]).T
print(len(X_pred))
```

```
2000
2284
```

In [ ]:
```
X_pred
```

Out[ ]:
```
array([[0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [1.00000000e+00, 1.00000000e-01, 1.66666667e-02],
       [2.00000000e+00, 2.00000000e-01, 3.33333333e-02],
       ...,
       [2.28100000e+03, 2.28100000e+02, 3.80166667e+01],
       [2.28200000e+03, 2.28200000e+02, 3.80333333e+01],
       [2.28300000e+03, 2.28300000e+02, 3.80500000e+01]])
```

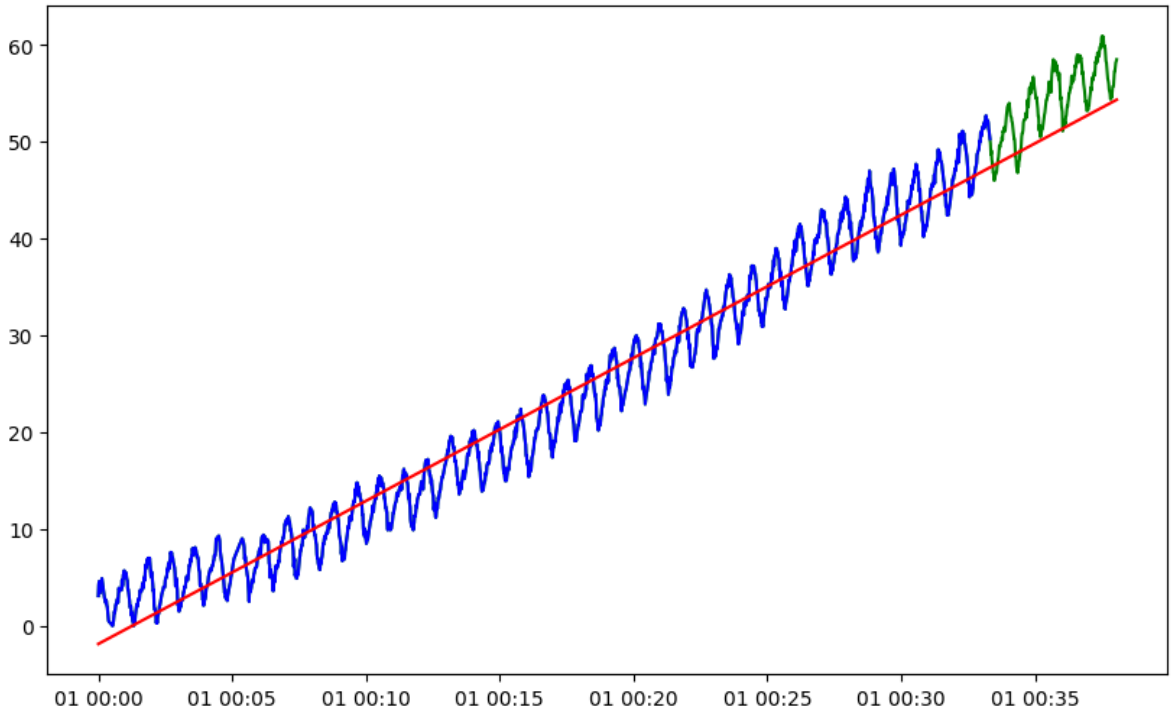
Da Random-Forest, normalerweise nicht mit Zeitreihen umgehen muss, müssen die Daten über Features dem Model überbracht werden. In dem Fall werden die Zeitwerte als Variablen übernommen und auf dieser Basis trainiert.

Lineare Regression

Bevor mit dem Random-Forest gestartet wird, wird eine Lineare Regression durchgeführt um auch diese einmal zu zeigen.

In [ ]:
```
from sklearn.linear_model import LinearRegression

# Model erstellen
model = LinearRegression()
model.fit(X_train, car_data_random_forest_train.values)

# Vorhersage
preds = model.predict(X_pred)

# Plot
plt.figure(figsize=(10, 6))
plt.plot(car_data_random_forest.index, car_data.values, color='green')
plt.plot(car_data_random_forest_train.index, car_data_random_forest_train.values ,
plt.plot(car_data_random_forest.index, preds , color='red')
```

Out[ ]: [<matplotlib.lines.Line2D at 0x20beceb74a0>]

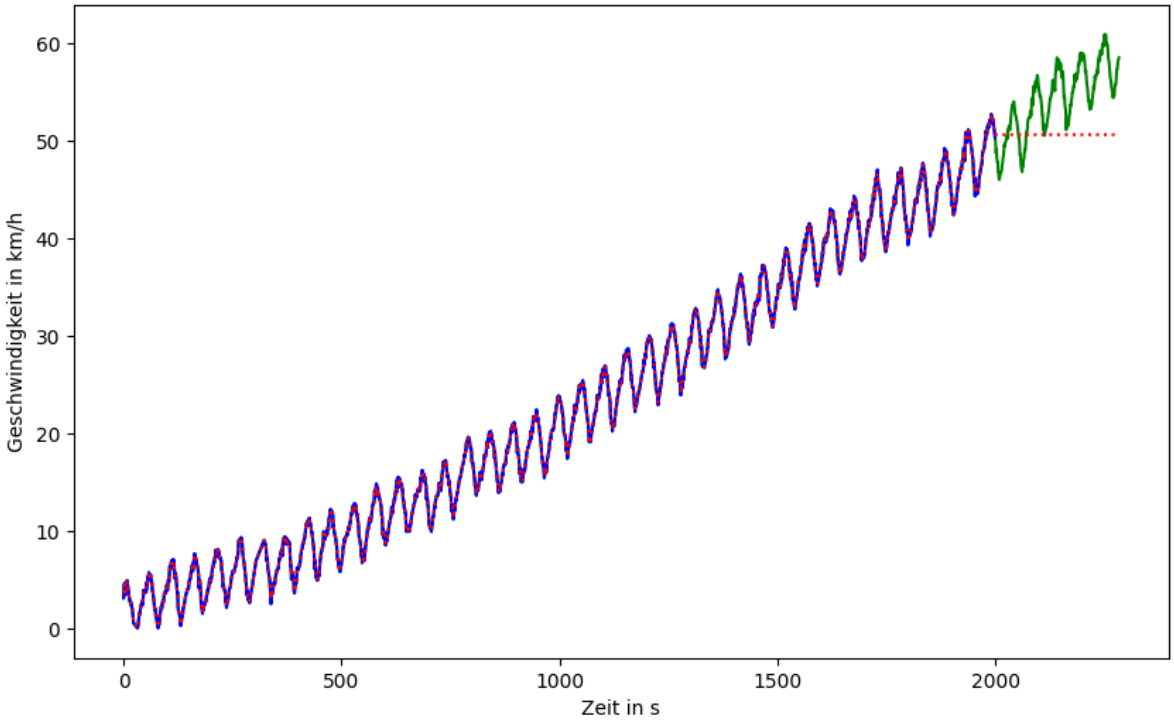


Auch hier sind in blau wieder die Trainings-Daten und in grün die Test-Daten. In diesem Fall wird jedoch über den gesamten Graph die Werte vorhergesagt. Es ist zu sehen, dass sich die Regression relativ gut an die Kurve anpasst.

Random Forest

In [ ]:
```
from datetime import datetime
from sklearn.ensemble import RandomForestRegressor

# Model erstellen
model = RandomForestRegressor()
train, test = train_test_split(car_data_random_forest.values, train_size=2200)
model.fit(X_train, car_data_random_forest_train.values)

# Vorhersage
preds = model.predict(X_pred)

# Plot
start_time = datetime(2022, 1, 1, 0, 0, 0)
seconds_since_start = (car_data_random_forest.index - start_time).total_seconds()
seconds_since_start_train = (car_data_random_forest_train.index - start_time).total
plt.figure(figsize=(10, 6))

plt.plot(seconds_since_start, car_data.values, color='green')
plt.plot(seconds_since_start_train, car_data_random_forest_train.values , color='bl
plt.plot(seconds_since_start, preds , color='red', linestyle=':')
plt.ylabel('Geschwindigkeit in km/h')
plt.xlabel('Zeit in s')
plt.savefig('pictures/Modellierung/RamdomForest.pdf')
```

```
plt.show()
```



Auch hier sind in blau wieder die Trainings-Daten und in grün die Test-Daten. In diesem Fall ist zu erkennen, dass der Random Forest sehr gut in der Vorhersage der Werte ist, die als Trainingsdaten vorlagen. Sobald es außerhalb des Bereiches der Trainingsdaten geht, bleibt die Vorhersage konstant. Das kann darauf deuten, dass es sich hier um overfitting handlen kann.

LSTM

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Input
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error

tf.random.set_seed(7)
```

In [ ]:
```
car_data_LSTM = car_data
df_LSTM = pd.DataFrame(car_data_LSTM)
df_LSTM

dataset_LSTM = df_LSTM.values
dataset_LSTM = dataset_LSTM.astype('float32')
dataset_LSTM
```

Out[ ]:
```
array([[ 3.1],
       [ 4.3],
       [ 4.6],
       ...,
       [58.2],
       [58.3],
       [58.5]], dtype=float32)
```

Für das umsetzten des LSTM müssen die Daten in das passende Format gebracht werden

In [ ]:
```
scaler = MinMaxScaler(feature_range=(0, 1))
dataset_LSTM = scaler.fit_transform(dataset_LSTM)
dataset_LSTM
```

Out[ ]:
```
array([[0.05090312],
       [0.07060755],
       [0.07553366],
       ...,
       [0.955665  ],
       [0.95730704],
       [0.9605911 ]], dtype=float32)
```

Damit das neuronale Netz die Werte besser verarbeiten kann werden sie davor skaliert, sodass alle Werte zwischen 0 und 1 liegen.

In [ ]:
```
train_size = int(len(dataset_LSTM) * 0.67)
test_size = len(dataset_LSTM) - train_size
train, test = dataset_LSTM[0:train_size,:], dataset_LSTM[train_size:len(dataset_LST
print(len(train), len(test))
```

```
1530 754
```

Die Daten werden prozentual in Trainings- und Test-Daten unterteilt. 67% sind dabei Trainings-Daten.

In [ ]:
```
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)
```

In [ ]:
```
look_back = 1
trainX, trainY = create_dataset(train, look_back)
testX, testY = create_dataset(test, look_back)
trainX = np.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
testX = np.reshape(testX, (testX.shape[0], 1, testX.shape[1]))
```

Weitere Schritte um die Daten für das Modell vorzubereiten.

In [ ]:
```
model = Sequential()
model.add(Input(shape=(1, look_back)))  # Input-Schicht mit der gewünschten Form (1
model.add(LSTM(4))
```

```
model.add(Dense(1))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(trainX, trainY, epochs=100, batch_size=1, verbose=2)
```

```
Epoch 1/100
1528/1528 - 1s - 799us/step - loss: 0.0093
Epoch 2/100
1528/1528 - 1s - 398us/step - loss: 1.1123e-04
Epoch 3/100
1528/1528 - 1s - 384us/step - loss: 1.0636e-04
Epoch 4/100
1528/1528 - 1s - 390us/step - loss: 9.7790e-05
Epoch 5/100
1528/1528 - 1s - 392us/step - loss: 8.9217e-05
Epoch 6/100
1528/1528 - 1s - 388us/step - loss: 8.3888e-05
Epoch 7/100
1528/1528 - 1s - 387us/step - loss: 8.1354e-05
Epoch 8/100
1528/1528 - 1s - 393us/step - loss: 8.0319e-05
Epoch 9/100
1528/1528 - 1s - 402us/step - loss: 7.9906e-05
Epoch 10/100
1528/1528 - 1s - 409us/step - loss: 7.9724e-05
Epoch 11/100
1528/1528 - 1s - 407us/step - loss: 7.9623e-05
Epoch 12/100
1528/1528 - 1s - 399us/step - loss: 7.9552e-05
Epoch 13/100
1528/1528 - 1s - 397us/step - loss: 7.9492e-05
Epoch 14/100
1528/1528 - 1s - 391us/step - loss: 7.9437e-05
Epoch 15/100
1528/1528 - 1s - 396us/step - loss: 7.9384e-05
Epoch 16/100
1528/1528 - 1s - 390us/step - loss: 7.9332e-05
Epoch 17/100
1528/1528 - 1s - 379us/step - loss: 7.9281e-05
Epoch 18/100
1528/1528 - 1s - 389us/step - loss: 7.9230e-05
Epoch 19/100
1528/1528 - 1s - 384us/step - loss: 7.9181e-05
Epoch 20/100
1528/1528 - 1s - 376us/step - loss: 7.9131e-05
Epoch 21/100
1528/1528 - 1s - 375us/step - loss: 7.9083e-05
Epoch 22/100
1528/1528 - 1s - 398us/step - loss: 7.9035e-05
Epoch 23/100
1528/1528 - 1s - 393us/step - loss: 7.8987e-05
Epoch 24/100
1528/1528 - 1s - 397us/step - loss: 7.8941e-05
Epoch 25/100
1528/1528 - 1s - 404us/step - loss: 7.8894e-05
Epoch 26/100
1528/1528 - 1s - 414us/step - loss: 7.8848e-05
Epoch 27/100
1528/1528 - 1s - 408us/step - loss: 7.8803e-05
Epoch 28/100
1528/1528 - 1s - 407us/step - loss: 7.8758e-05
```

```
Epoch 29/100
1528/1528 - 1s - 398us/step - loss: 7.8714e-05
Epoch 30/100
1528/1528 - 1s - 395us/step - loss: 7.8670e-05
Epoch 31/100
1528/1528 - 1s - 389us/step - loss: 7.8626e-05
Epoch 32/100
1528/1528 - 1s - 380us/step - loss: 7.8583e-05
Epoch 33/100
1528/1528 - 1s - 385us/step - loss: 7.8541e-05
Epoch 34/100
1528/1528 - 1s - 386us/step - loss: 7.8499e-05
Epoch 35/100
1528/1528 - 1s - 404us/step - loss: 7.8458e-05
Epoch 36/100
1528/1528 - 1s - 374us/step - loss: 7.8416e-05
Epoch 37/100
1528/1528 - 1s - 374us/step - loss: 7.8376e-05
Epoch 38/100
1528/1528 - 1s - 384us/step - loss: 7.8336e-05
Epoch 39/100
1528/1528 - 1s - 373us/step - loss: 7.8296e-05
Epoch 40/100
1528/1528 - 1s - 374us/step - loss: 7.8257e-05
Epoch 41/100
1528/1528 - 1s - 376us/step - loss: 7.8218e-05
Epoch 42/100
1528/1528 - 1s - 380us/step - loss: 7.8180e-05
Epoch 43/100
1528/1528 - 1s - 397us/step - loss: 7.8142e-05
Epoch 44/100
1528/1528 - 1s - 373us/step - loss: 7.8105e-05
Epoch 45/100
1528/1528 - 1s - 387us/step - loss: 7.8068e-05
Epoch 46/100
1528/1528 - 1s - 386us/step - loss: 7.8031e-05
Epoch 47/100
1528/1528 - 1s - 386us/step - loss: 7.7995e-05
Epoch 48/100
1528/1528 - 1s - 388us/step - loss: 7.7959e-05
Epoch 49/100
1528/1528 - 1s - 386us/step - loss: 7.7924e-05
Epoch 50/100
1528/1528 - 1s - 387us/step - loss: 7.7889e-05
Epoch 51/100
1528/1528 - 1s - 400us/step - loss: 7.7855e-05
Epoch 52/100
1528/1528 - 1s - 395us/step - loss: 7.7821e-05
Epoch 53/100
1528/1528 - 1s - 386us/step - loss: 7.7787e-05
Epoch 54/100
1528/1528 - 1s - 384us/step - loss: 7.7754e-05
Epoch 55/100
1528/1528 - 1s - 387us/step - loss: 7.7721e-05
Epoch 56/100
1528/1528 - 1s - 366us/step - loss: 7.7688e-05
```

```
Epoch 57/100
1528/1528 - 1s - 373us/step - loss: 7.7656e-05
Epoch 58/100
1528/1528 - 1s - 386us/step - loss: 7.7624e-05
Epoch 59/100
1528/1528 - 1s - 401us/step - loss: 7.7593e-05
Epoch 60/100
1528/1528 - 1s - 408us/step - loss: 7.7562e-05
Epoch 61/100
1528/1528 - 1s - 399us/step - loss: 7.7531e-05
Epoch 62/100
1528/1528 - 1s - 392us/step - loss: 7.7501e-05
Epoch 63/100
1528/1528 - 1s - 389us/step - loss: 7.7471e-05
Epoch 64/100
1528/1528 - 1s - 390us/step - loss: 7.7442e-05
Epoch 65/100
1528/1528 - 1s - 389us/step - loss: 7.7413e-05
Epoch 66/100
1528/1528 - 1s - 394us/step - loss: 7.7384e-05
Epoch 67/100
1528/1528 - 1s - 395us/step - loss: 7.7356e-05
Epoch 68/100
1528/1528 - 1s - 397us/step - loss: 7.7327e-05
Epoch 69/100
1528/1528 - 1s - 392us/step - loss: 7.7300e-05
Epoch 70/100
1528/1528 - 1s - 386us/step - loss: 7.7272e-05
Epoch 71/100
1528/1528 - 1s - 381us/step - loss: 7.7245e-05
Epoch 72/100
1528/1528 - 1s - 390us/step - loss: 7.7218e-05
Epoch 73/100
1528/1528 - 1s - 385us/step - loss: 7.7192e-05
Epoch 74/100
1528/1528 - 1s - 385us/step - loss: 7.7166e-05
Epoch 75/100
1528/1528 - 1s - 397us/step - loss: 7.7140e-05
Epoch 76/100
1528/1528 - 1s - 399us/step - loss: 7.7115e-05
Epoch 77/100
1528/1528 - 1s - 396us/step - loss: 7.7090e-05
Epoch 78/100
1528/1528 - 1s - 392us/step - loss: 7.7065e-05
Epoch 79/100
1528/1528 - 1s - 383us/step - loss: 7.7041e-05
Epoch 80/100
1528/1528 - 1s - 378us/step - loss: 7.7017e-05
Epoch 81/100
1528/1528 - 1s - 374us/step - loss: 7.6993e-05
Epoch 82/100
1528/1528 - 1s - 376us/step - loss: 7.6969e-05
Epoch 83/100
1528/1528 - 1s - 373us/step - loss: 7.6946e-05
Epoch 84/100
1528/1528 - 1s - 374us/step - loss: 7.6923e-05
```

```
Epoch 85/100
1528/1528 - 1s - 372us/step - loss: 7.6901e-05
Epoch 86/100
1528/1528 - 1s - 395us/step - loss: 7.6878e-05
Epoch 87/100
1528/1528 - 1s - 387us/step - loss: 7.6856e-05
Epoch 88/100
1528/1528 - 1s - 387us/step - loss: 7.6835e-05
Epoch 89/100
1528/1528 - 1s - 393us/step - loss: 7.6813e-05
Epoch 90/100
1528/1528 - 1s - 392us/step - loss: 7.6792e-05
Epoch 91/100
1528/1528 - 1s - 391us/step - loss: 7.6771e-05
Epoch 92/100
1528/1528 - 1s - 391us/step - loss: 7.6750e-05
Epoch 93/100
1528/1528 - 1s - 382us/step - loss: 7.6730e-05
Epoch 94/100
1528/1528 - 1s - 417us/step - loss: 7.6710e-05
Epoch 95/100
1528/1528 - 1s - 405us/step - loss: 7.6691e-05
Epoch 96/100
1528/1528 - 1s - 410us/step - loss: 7.6671e-05
Epoch 97/100
1528/1528 - 1s - 418us/step - loss: 7.6652e-05
Epoch 98/100
1528/1528 - 1s - 409us/step - loss: 7.6633e-05
Epoch 99/100
1528/1528 - 1s - 405us/step - loss: 7.6614e-05
Epoch 100/100
1528/1528 - 1s - 405us/step - loss: 7.6596e-05
```

Out[ ]: `<keras.src.callbacks.history.History at 0x20bfba7c740>`

Das Model wird nun aus verschiedenen Schichten zusammengestellt. Es liegt ein LST und ein Dense Layer vor. Eine Optimierung erfolgt dann noch mit dem Optimizer Adam.

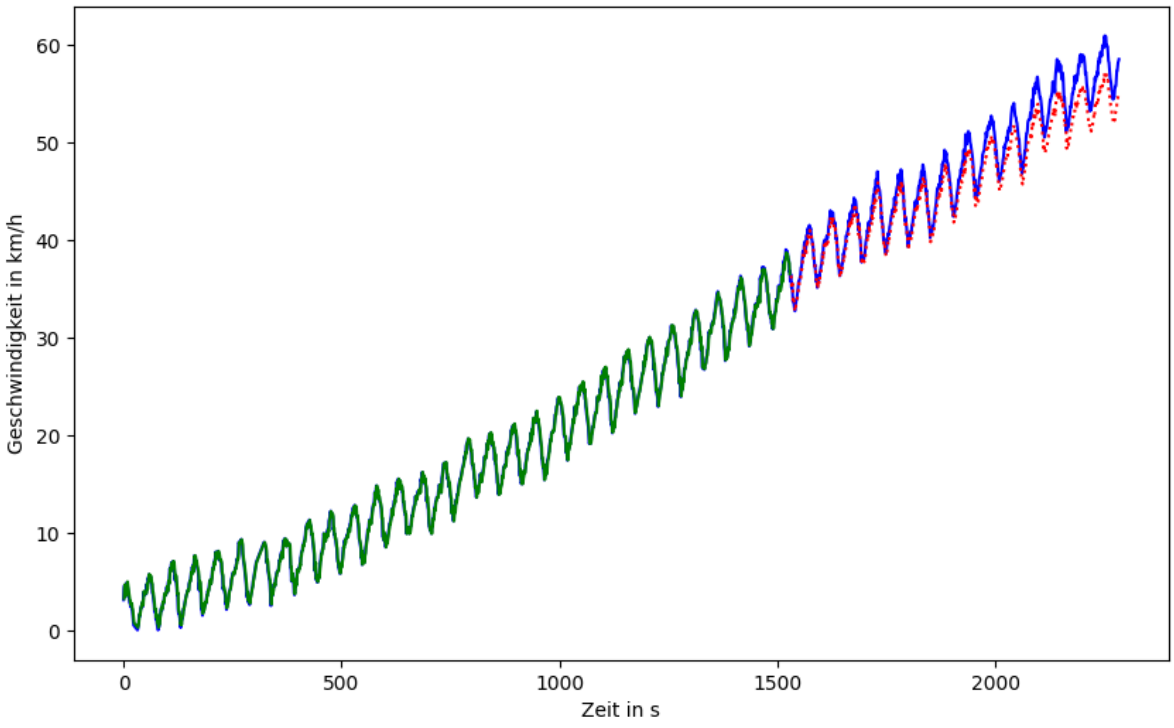
In [ ]:
```
trainPredict = model.predict(trainX)
testPredict = model.predict(testX)

trainPredict = scaler.inverse_transform(trainPredict)
trainY = scaler.inverse_transform([trainY])
testPredict = scaler.inverse_transform(testPredict)
testY = scaler.inverse_transform([testY])

trainScore = np.sqrt(mean_squared_error(trainY[0], trainPredict[:,0]))
print('Train Score: %.2f RMSE' % (trainScore))
testScore = np.sqrt(mean_squared_error(testY[0], testPredict[:,0]))
print('Test Score: %.2f RMSE' % (testScore))
```

```
48/48 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 450us/step
Train Score: 0.47 RMSE
Test Score: 1.73 RMSE
```

Nachdem das Model erstellt wurde, werden Vorhersagen gemacht. Damit die Daten auch wieder interpretiert werden können, müssen diese wieder umgeformt werden, damit sie nicht mehr zwischen 0 und 1 liegen.

Auch die Fehlerwerte für die Trainings- und Test-Daten haben mit 0.5 und 1.1 sehr gut abgeschnitten.

In [ ]:
```
trainPredictPlot = np.empty_like(dataset_LSTM)
trainPredictPlot[:, :] = np.nan
trainPredictPlot[look_back:len(trainPredict)+look_back, :] = trainPredict
testPredictPlot = np.empty_like(dataset_LSTM)
testPredictPlot[:, :] = np.nan
testPredictPlot[len(trainPredict)+(look_back*2)+1:len(dataset_LSTM)-1, :] = testPre

# plot baseline and predictions
plt.figure(figsize=(10, 6))
plt.plot(scaler.inverse_transform(dataset_LSTM), color='blue')
plt.plot(trainPredictPlot, color='green')
plt.plot(testPredictPlot, linestyle=':',    color='red')
plt.ylabel('Geschwindigkeit in km/h')
plt.xlabel('Zeit in s')
plt.savefig('pictures/Modellierung/LSTM.pdf')
plt.show()
```



In blau sind hierbei die tatsächlichen Werte zu sehen. In grün sind die Vorhersagen, die auf Basis des Trainings-Daten gemacht wurde. Diese liegen ziemlich genau über dem blauen Graphen weshalb man hier fast nur den grünen sieht. In rot gepunktet, sind die vorhergesagten Daten zu sehen, welche nicht zum Trainieren verwendet wurden. In beiden Fällen ist ein sehr gutes Ergebnis zu beobachten. Bei der roten Linie ist zu erkennen umso weiter sie sich von den Trainings-Daten entfernen umso ungenauer werden sie. Dennoch

sind auch gegen Ende noch sehr gute Werte zu erkennen.