Kiosk24

October 26, 2023

Kiosk24

Late-night convenience stores, popularly known as Spaeti (German trivialization for late-night stores) have a cult status in Germany and have historically (together with Gas-stations) catered the needs for late-night groceries and other essential suppies when normal supermarkets are closed. Kiosk24 is Germany's largest all-day convenience store chain and is the leader in the country's 4 billion Euro late-night convinience store market. Their USP is that they are, unlike regular shops, open round-the-clock and 24/7.

0.0.1 Problem statement

Kiosk24 started a pilot project in Berlin offering bicycle based delivery services. It has been gathering data over a period of one year and wants to get an estimated delivery time that it can provide the customers on the basis of a 10 variable strong feature-set that includes among others the distance from the store to their location, total on-duty delivery partners, no. of items ordered and total outstanding orders.

This dataset hence has the required data to train a regression model that will estimate the delivery-time (dependent variable) based on the feature-set (independent variable).

0.1 Feature Set

store_id: individual id of the store

created_at: the timestamp at which the order was placed

delivered_at: the timestamp when the order was delivered

total_items: total number of items

distinct_items: number of distinct items in the order

subtotal: final bill of the order

total_onshift_runners: no. of delivery partners on duty at order placement

total_busy_runners: number of delivery partners attending to other tasks

total_outstanding_orders: total number of orders to be fulfilled at the moment

store_to_consumer_driving_duration_seconds: approximate travel time from restaurant to customer

0.2 ML Development Lifecycle

0.2.1 Data preperation

importing libraries

[39]:
```
import warnings
warnings.filterwarnings("ignore")
```

[2]:
```
#for reading and handling the data
import pandas as pd
import numpy as np
import os

#for visualizinng and analyzing data
import matplotlib.pyplot as plt
import seaborn as sns

#data preprocessing
from sklearn.neighbors import LocalOutlierFactor
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

#random forest model training
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_error
from sklearn.ensemble import RandomForestRegressor

#artificial neural network training
from sklearn import preprocessing
from tensorflow.keras import Model
from tensorflow.keras import Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.layers import Dense,Dropout,BatchNormalization,LeakyReLU
from sklearn.model_selection import train_test_split
from tensorflow.keras.losses import MeanSquaredLogarithmicError
from tensorflow.keras.losses import MeanSquaredError
from tensorflow.keras.losses import MeanAbsolutePercentageError
from tensorflow.keras.metrics import mean_absolute_percentage_error
from tensorflow.keras.metrics import RootMeanSquaredError
from tensorflow.keras.metrics import MeanAbsoluteError
from tensorflow.keras.optimizers import SGD,Adam
from sklearn.metrics import mean_absolute_percentage_error
from tensorflow.keras.models import load_model

#Modelbit model deployment
import modelbit
```

```
import joblib
```

[3]:
```
sns.set(rc={'figure.figsize':(11.7,8.27)})
```

loading data

[4]:
```
for dirname, _, filenames in os.walk('C:\Dataset'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

[5]:
```
df=pd.read_excel('Kiosk24_data.xlsx')
```

understanding data as seen by programming environment

[6]:
```
df.head()
```

[6]:
```
   store_id          created_at        delivered_at  total_items  \
0         1 2015-02-06 22:24:00 2015-02-06 23:11:00            4
1         2 2015-02-10 21:49:00 2015-02-10 22:33:00            1
2         2 2015-02-16 00:11:00 2015-02-16 01:06:00            4
3         1 2015-02-12 03:36:00 2015-02-12 04:35:00            1
4         1 2015-01-27 02:12:00 2015-01-27 02:58:00            2

   distinct_items  subtotal  total_onshift_runners  total_busy_runners  \
0               4     39.10                     33                  14
1               1     21.59                      1                   2
2               3     54.22                      8                   6
3               1     17.33                      5                   6
4               2     41.14                      5                   5

   total_outstanding_orders  store_to_consumer_driving_duration_seconds
0                        21                                        861
1                         2                                        690
2                        18                                        289
3                         8                                        795
4                         7                                        205
```

[7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 175777 entries, 0 to 175776
Data columns (total 10 columns):
 #   Column                                    Non-Null Count   Dtype
---  ------                                    --------------   -----
 0   store_id                                  175777 non-null  int64
 1   created_at                                175777 non-null  datetime64[ns]
 2   delivered_at                              175777 non-null  datetime64[ns]
 3   total_items                               175777 non-null  int64
 4   distinct_items                            175777 non-null  int64
```

```
 5   subtotal                                  175777 non-null  float64
 6   total_onshift_runners                     175777 non-null  int64
 7   total_busy_runners                        175777 non-null  int64
 8   total_outstanding_orders                  175777 non-null  int64
 9   store_to_consumer_driving_duration_seconds  175777 non-null  int64
dtypes: datetime64[ns](2), float64(1), int64(7)
memory usage: 13.4 MB
```

0.2.2 Feature engineering

preparing the data for further analysis

[8]:
```
df['store_to_consumer_driving_duration_mins']=round(df['store_to_consumer_driving_duration_sec
 ↪60, 0)
```

[9]:
```
df['time_to_delivery']=df['delivered_at'] - df['created_at']
```

[10]:
```
df['time_to_delivery_mins']=df['time_to_delivery']/pd.Timedelta('60s')
```

[11]:
```
df['created_at_hour']=df['created_at'].dt.hour
df['created_at_day']=df['created_at'].dt.dayofweek
```

[12]:
```
# Define a dictionary to map numeric day of the week to day names
day_names = {
    0: 'Monday',
    1: 'Tuesday',
    2: 'Wednesday',
    3: 'Thursday',
    4: 'Friday',
    5: 'Saturday',
    6: 'Sunday'
}
```

[13]:
```
# Create a new column with day names based on the 'created_at_day' column
df['created_at_day_name'] = df['created_at_day'].map(day_names)
```

cleaning data

[14]:
```
df.
 ↪drop(['store_to_consumer_driving_duration_seconds','time_to_delivery','created_at','deliver
```

[15]:
```
df.isna().sum()
```

[15]:
```
store_id                    0
total_items                 0
distinct_items              0
subtotal                    0
total_onshift_runners       0
total_busy_runners          0
```

```
total_outstanding_orders                    0
store_to_consumer_driving_duration_mins     0
time_to_delivery_mins                       0
created_at_hour                             0
created_at_day                              0
created_at_day_name                         0
dtype: int64
```

[16]:
```
df.dropna(inplace=True)
```

[17]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 175777 entries, 0 to 175776
Data columns (total 12 columns):
 #   Column                                  Non-Null Count   Dtype
---  ------                                  --------------   -----
 0   store_id                                175777 non-null  int64
 1   total_items                             175777 non-null  int64
 2   distinct_items                          175777 non-null  int64
 3   subtotal                                175777 non-null  float64
 4   total_onshift_runners                   175777 non-null  int64
 5   total_busy_runners                      175777 non-null  int64
 6   total_outstanding_orders                175777 non-null  int64
 7   store_to_consumer_driving_duration_mins 175777 non-null  float64
 8   time_to_delivery_mins                   175777 non-null  float64
 9   created_at_hour                         175777 non-null  int64
 10  created_at_day                          175777 non-null  int64
 11  created_at_day_name                     175777 non-null  object
dtypes: float64(3), int64(8), object(1)
memory usage: 16.1+ MB
```

[18]:
```
df['store_id']=df['store_id'].astype('category').cat.codes
```

[19]:
```
df['created_at_hour']=df['created_at_hour'].astype('category').cat.codes
```

[20]:
```
df['created_at_day']=df['created_at_day'].astype('category').cat.codes
```

[21]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 175777 entries, 0 to 175776
Data columns (total 12 columns):
 #   Column                                  Non-Null Count   Dtype
---  ------                                  --------------   -----
 0   store_id                                175777 non-null  int8
 1   total_items                             175777 non-null  int64
 2   distinct_items                          175777 non-null  int64
```

```
 3   subtotal                                  175777 non-null  float64
 4   total_onshift_runners                     175777 non-null  int64
 5   total_busy_runners                        175777 non-null  int64
 6   total_outstanding_orders                  175777 non-null  int64
 7   store_to_consumer_driving_duration_mins   175777 non-null  float64
 8   time_to_delivery_mins                     175777 non-null  float64
 9   created_at_hour                           175777 non-null  int8
 10  created_at_day                            175777 non-null  int8
 11  created_at_day_name                       175777 non-null  object
dtypes: float64(3), int64(5), int8(3), object(1)
memory usage: 12.6+ MB
```

[22]:
```
df.head()
```

[22]:

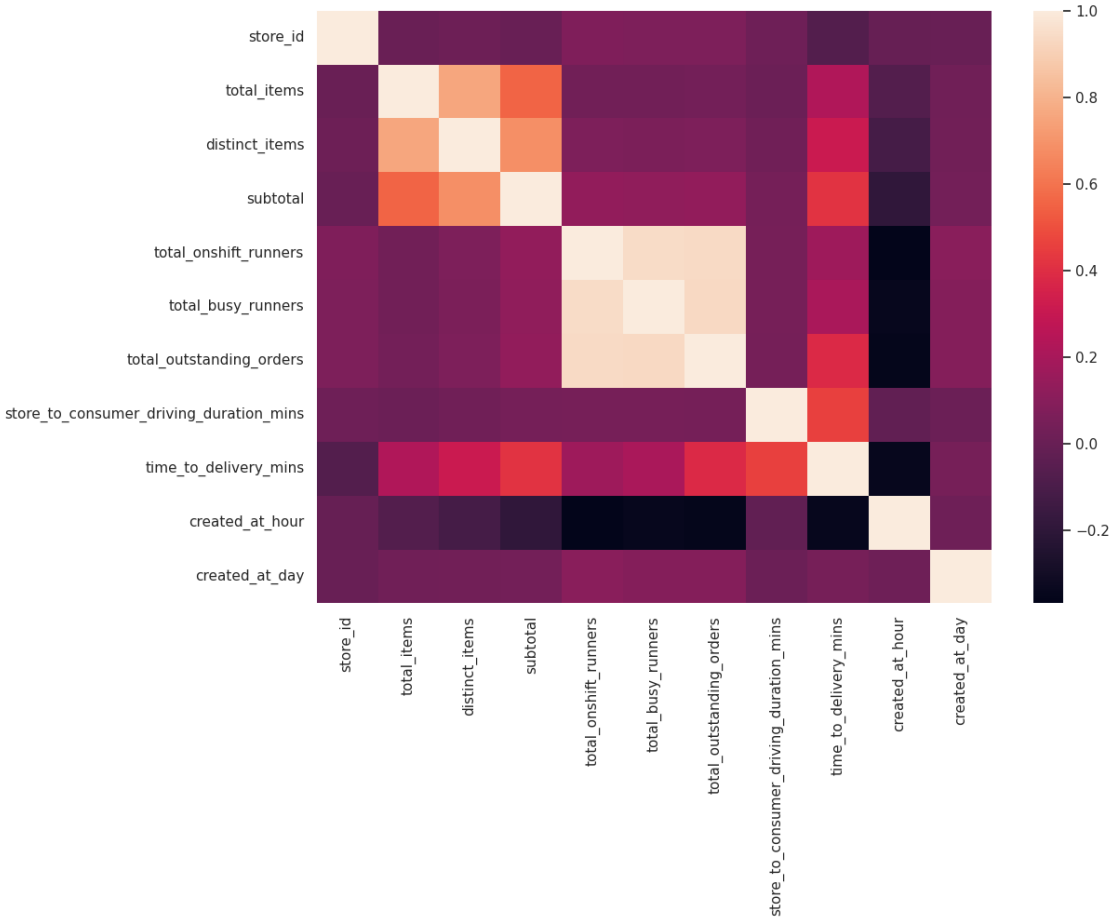
	store_id	total_items	distinct_items	subtotal	total_onshift_runners	total_busy_runners	total_outstanding_orders	store_to_consumer_driving_duration_mins	time_to_delivery_mins	created_at_hour	created_at_day	created_at_day_name
0	0	4	4	39.10	33	14	21	14.0	47.0	17	4	Friday
1	1	1	1	21.59	1	2	2	12.0	44.0	16	1	Tuesday
2	1	4	3	54.22	8	6	18	5.0	55.0	0	0	Monday
3	0	1	1	17.33	5	6	8	13.0	59.0	3	3	Thursday
4	0	2	2	41.14	5	5	7	3.0	46.0	2	1	Tuesday

[23]:
```
#df.to_excel("Kiosk24_data_mod.xlsx", index=False)
```

visualizing data

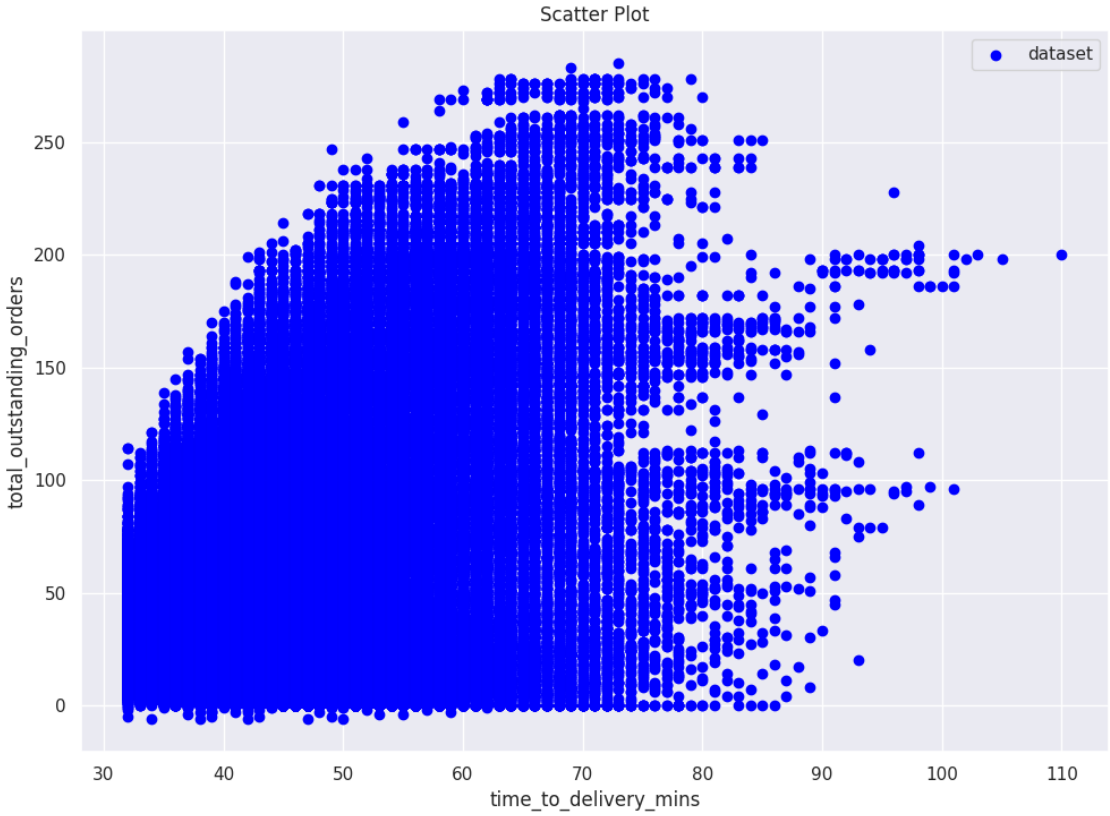
[24]:
```
sns.heatmap(df[[col for col in df.columns if col != 'created_at_day_name']].
↪corr())
```

[24]: <Axes: >



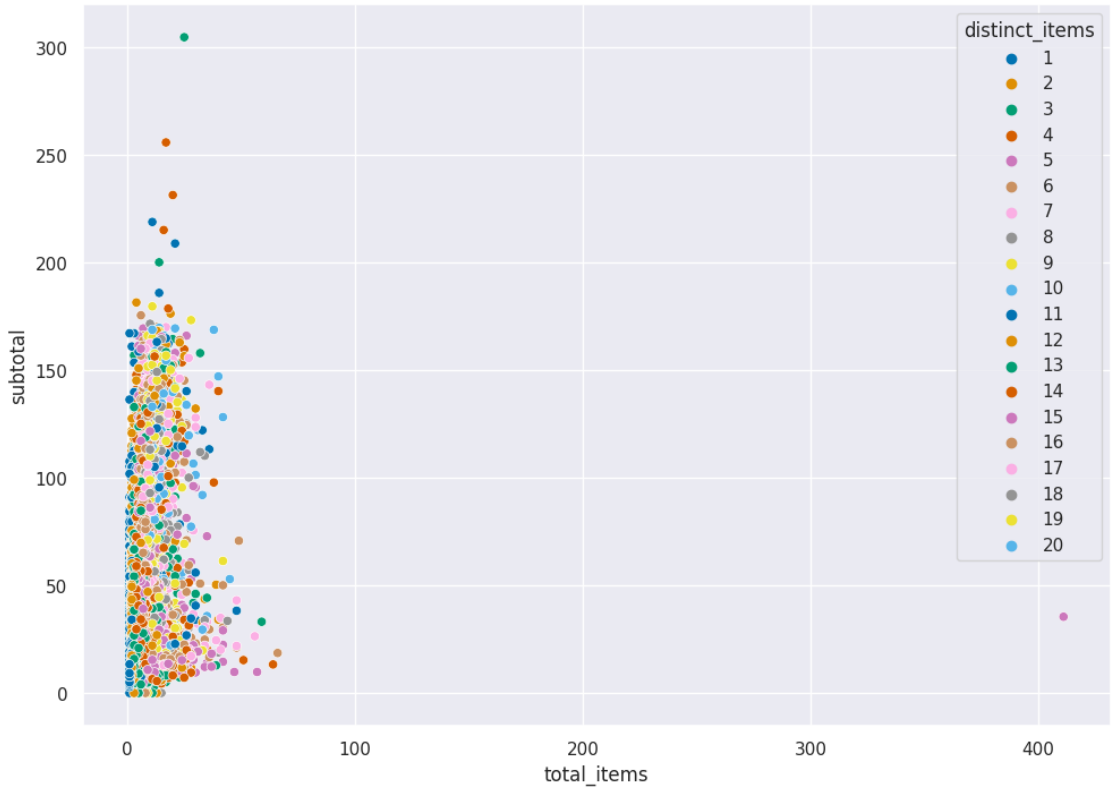
[25]:
```
plt.scatter(df.time_to_delivery_mins, df.total_outstanding_orders,␣
 ↪label='dataset', color='blue')

plt.xlabel('time_to_delivery_mins')
plt.ylabel('total_outstanding_orders')
plt.title('Scatter Plot')

# Add a legend to distinguish the two DataFrames
plt.legend()
```

[25]: <matplotlib.legend.Legend at 0x798f07398dc0>



[26]:
```
sns.
↪scatterplot(x='total_items',y='subtotal',hue='distinct_items',palette='colorblind',data=df)
```

[26]: <Axes: xlabel='total_items', ylabel='subtotal'>



identifying outliers

[27]:
```
model1=LocalOutlierFactor()
#model1.fit(df)
df['lof_anomaly_score']=model1.fit_predict(df[[col for col in df.columns if col␣
 ↪!= 'created_at_day_name']])
```

[28]:
```
print("number of outliers : ",(len(df.loc[(df['lof_anomaly_score'] == -1)])))
df_anomaly=df.loc[(df['lof_anomaly_score'] == -1)]
df=df.loc[(df['lof_anomaly_score'] == 1)]
```

```
number of outliers :  801
```

[29]:
```
df.drop(['lof_anomaly_score'],axis=1,inplace=True)
```

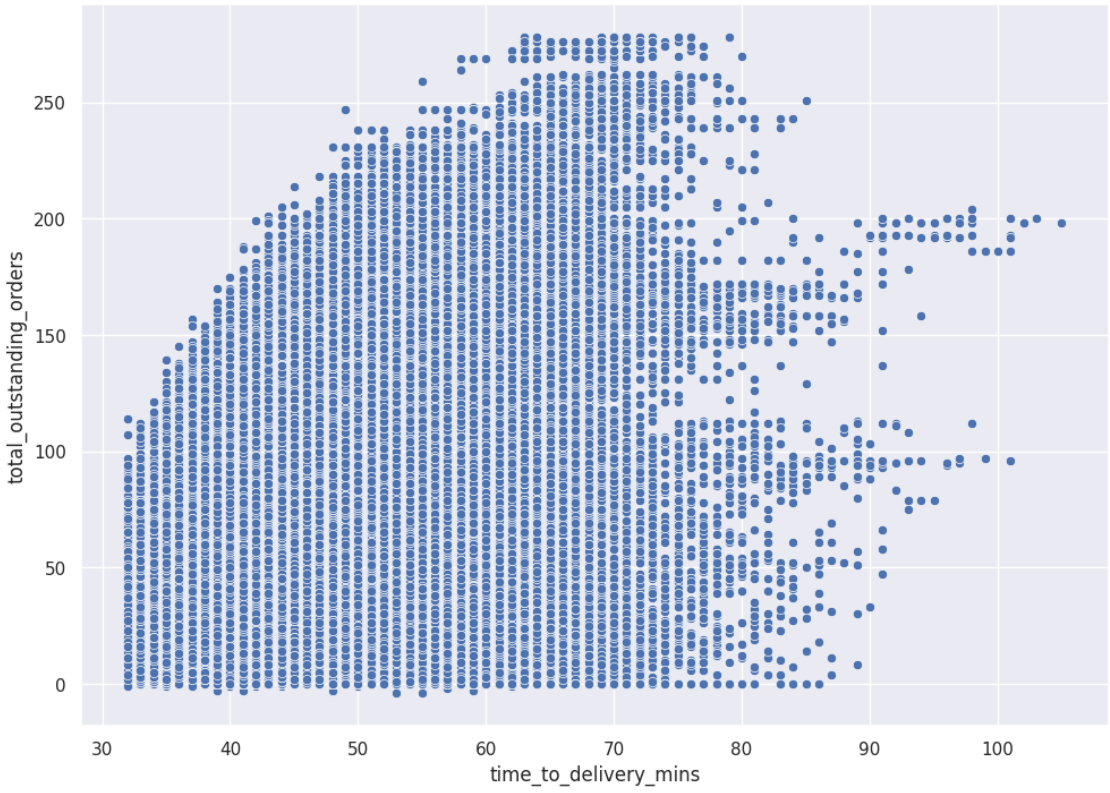
```
<ipython-input-29-a629ff9cb1da>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df.drop(['lof_anomaly_score'],axis=1,inplace=True)
```

[30]:
```
plt.scatter(df.time_to_delivery_mins, df.total_outstanding_orders,␣
 ↪label='Non-anomalous data', color='blue')
plt.scatter(df_anomaly.time_to_delivery_mins, df_anomaly.
 ↪total_outstanding_orders, label='Anomalous data', color='red')

plt.xlabel('time_to_delivery_mins')
plt.ylabel('total_outstanding_orders')
plt.title('Scatter Plot')

# Add a legend to distinguish the two DataFrames
plt.legend()
```

[30]: <matplotlib.legend.Legend at 0x798f0ba5a2f0>

[31]:
```
sns.scatterplot(x='time_to_delivery_mins',y='total_outstanding_orders',data=df)
```

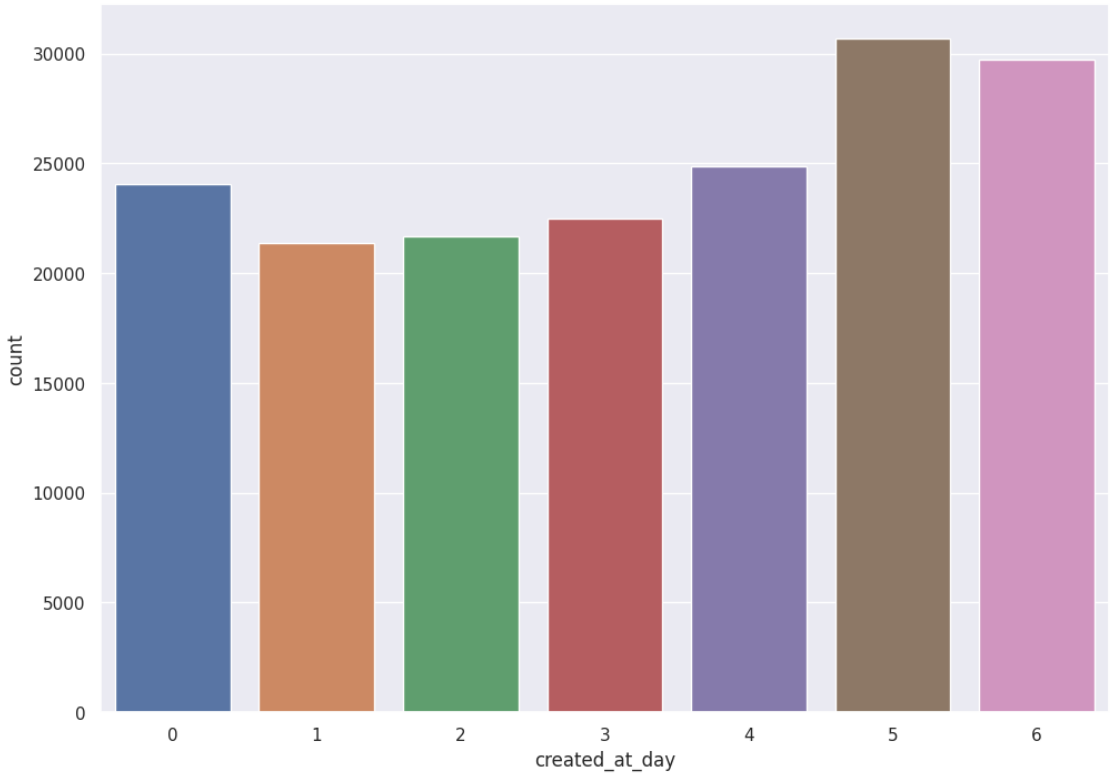
[31]: <Axes: xlabel='time_to_delivery_mins', ylabel='total_outstanding_orders'>



analyzing data

[32]:
```
sns.countplot(x=df.created_at_day)
```

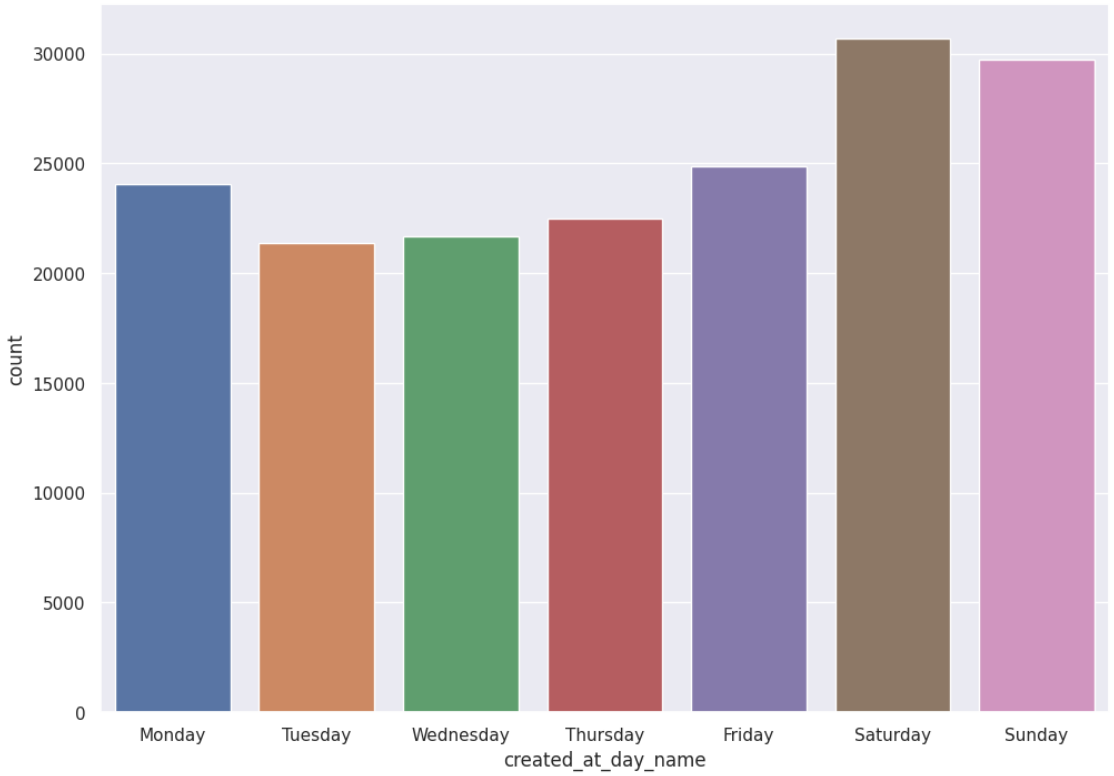
[32]:
```
<Axes: xlabel='created_at_day', ylabel='count'>
```



[33]:
```
# Specify the order of the x-axis categories
order = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday',␣
 ↪'Sunday']
```

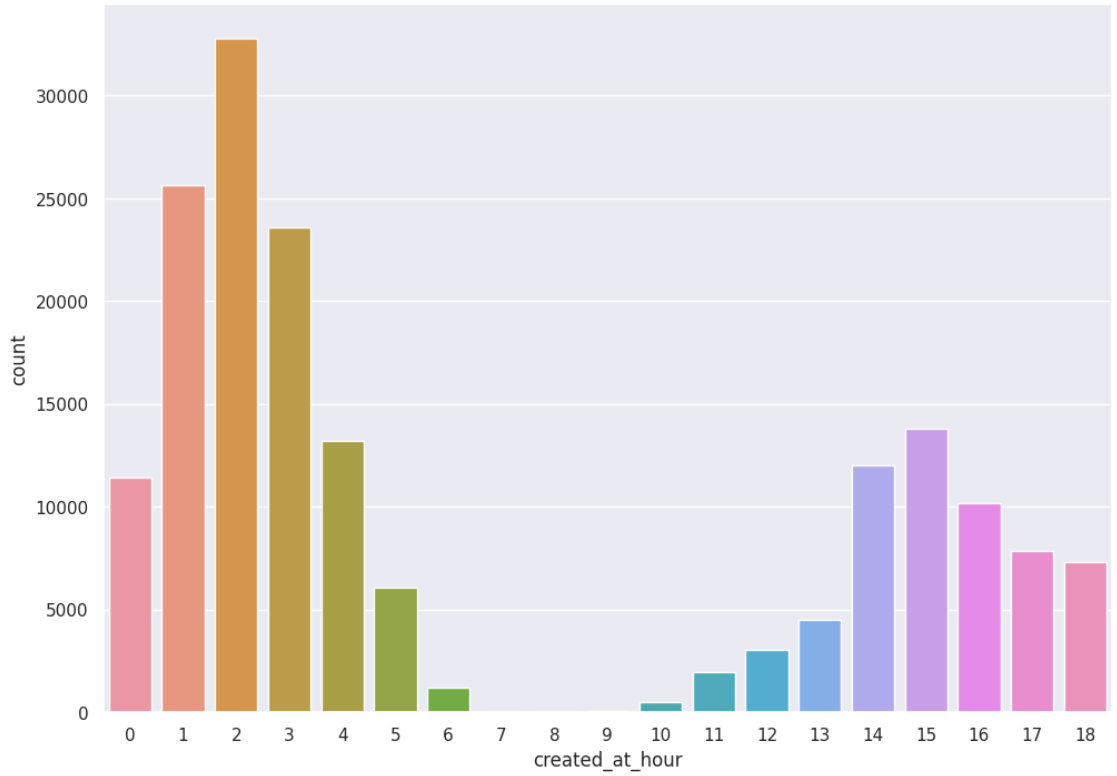
[34]:
```
sns.countplot(x=df.created_at_day_name, order=order)
```

[34]:
```
<Axes: xlabel='created_at_day_name', ylabel='count'>
```



[35]:
```
sns.countplot(x=df.created_at_hour)
```

[35]:
```
<Axes: xlabel='created_at_hour', ylabel='count'>
```



0.2.3 Data Splitting and Modelling

storing independent and dependent variables seperately

[36]:
```
y=df['time_to_delivery_mins']
x=df.drop(['time_to_delivery_mins', 'created_at_day_name'],axis=1)
```

[37]:
```
X_train_RF,X_test_RF,y_train_RF,y_test_RF=train_test_split(x,y,test_size=0.
↪2,random_state=42)
```

[38]:
```
x.head()
```

[38]:

	store_id	total_items	distinct_items	subtotal	total_onshift_runners	\
0	0	4	4	39.10	33	
1	1	1	1	21.59	1	
2	1	4	3	54.22	8	
3	0	1	1	17.33	5	
4	0	2	2	41.14	5	

	total_busy_runners	total_outstanding_orders	\
0	14	21	
1	2	2	
2	6	18	

```
3                    6                          8
4                    5                          7
```

```
   store_to_consumer_driving_duration_mins  created_at_hour  created_at_day
0                                     14.0               17               4
1                                     12.0               16               1
2                                      5.0                0               0
3                                     13.0                3               3
4                                      3.0                2               1
```

[40]:
```
y.head()
```

[40]:
```
0    47.0
1    44.0
2    55.0
3    59.0
4    46.0
Name: time_to_delivery_mins, dtype: float64
```

Supervised Learning - Random Forest Regression Model ***initializing the RF model***

[130]:
```
RF_regressor=RandomForestRegressor()
RF_regressor.fit(X_train_RF,y_train_RF)
```

[130]:
```
RandomForestRegressor()
```

training the RF model

[ ]:
```
prediction=RF_regressor.predict(X_test_RF)
```

calculating performance metrics

[131]:
```
mse=mean_squared_error(y_test_RF,prediction)
rmse=mse**.5
print("mse : ",mse)
print("rmse : ",rmse)
mae=mean_absolute_error(y_test_RF,prediction)
print("mase : ",mae)
```

```
mse :  3.9588216395423164
rmse :  1.9896787779795804
mase :  1.4754132581138526
```

[132]:
```
r2_score(y_test_RF,prediction)
```

[132]:
```
0.954812729345795
```

[133]:
```
def MAPE(Y_actual,Y_Predicted):
    mape=np.mean(np.abs((Y_actual - Y_Predicted)/Y_actual))*100
```

```
    return mape
```

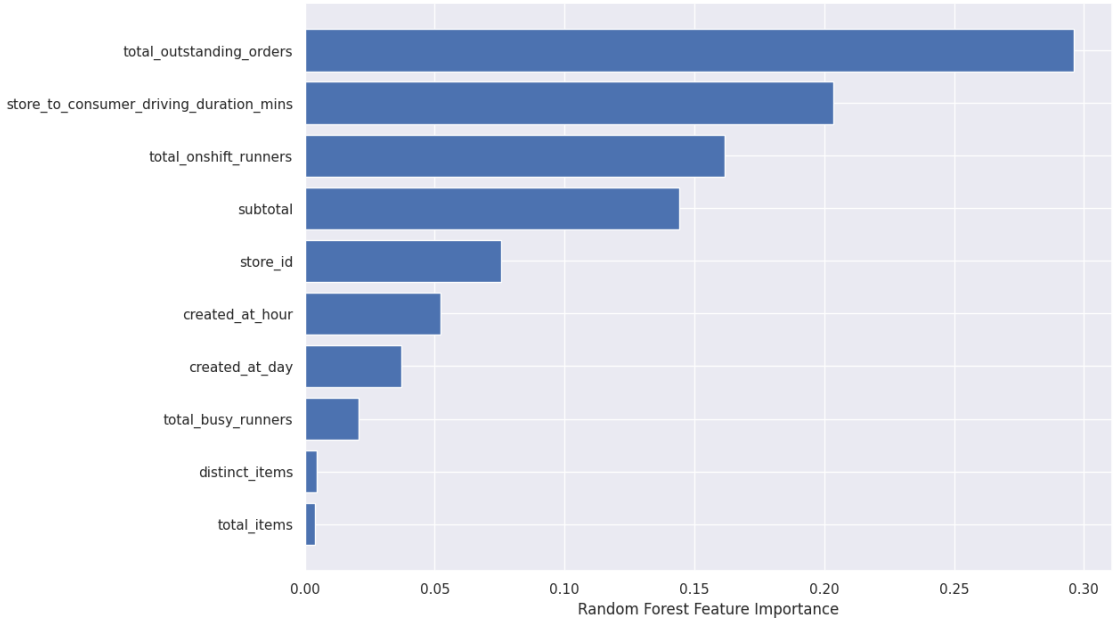
[134]:
```
print("mape : ",MAPE(y_test_RF,prediction))
```

```
mape :  3.232319290341489
```

plotting feature-importance metrics

[135]:
```
sorted_idx=RF_regressor.feature_importances_.argsort()
plt.barh(x.columns[sorted_idx],RF_regressor.feature_importances_[sorted_idx])
plt.xlabel("Random Forest Feature Importance")
```

[135]:
```
Text(0.5, 0, 'Random Forest Feature Importance')
```



Supervised Learning - Neural Networks ***scaling data before feeding to ANN***

[41]:
```
scaler=preprocessing.MinMaxScaler()
x_scaled=scaler.fit_transform(x)
X_train_ANN_scaled,X_test_ANN_scaled,y_train_ANN_scaled,y_test_ANN_scaled=train_test_split(x_s
  ↪2,random_state=42)
```

configuring the ANN

[42]:
```
ANN_model=Sequential()
ANN_model.add(Dense(14,kernel_initializer='normal',activation='relu'))
ANN_model.add(Dense(512,activation='relu'))
ANN_model.add(Dense(1024,activation='relu'))
ANN_model.add(Dense(256,activation='relu'))
```

```
ANN_model.add(Dense(1,activation='linear'))
```

training the ANN

[43]:
```
adam=Adam(learning_rate=0.01)
ANN_model.compile(loss='mse',optimizer=adam,metrics=['mse','mae'])
history=ANN_model.
  ↪fit(X_train_ANN_scaled,y_train_ANN_scaled,epochs=30,batch_size=512,verbose=1,validation_spl
  ↪2)
```

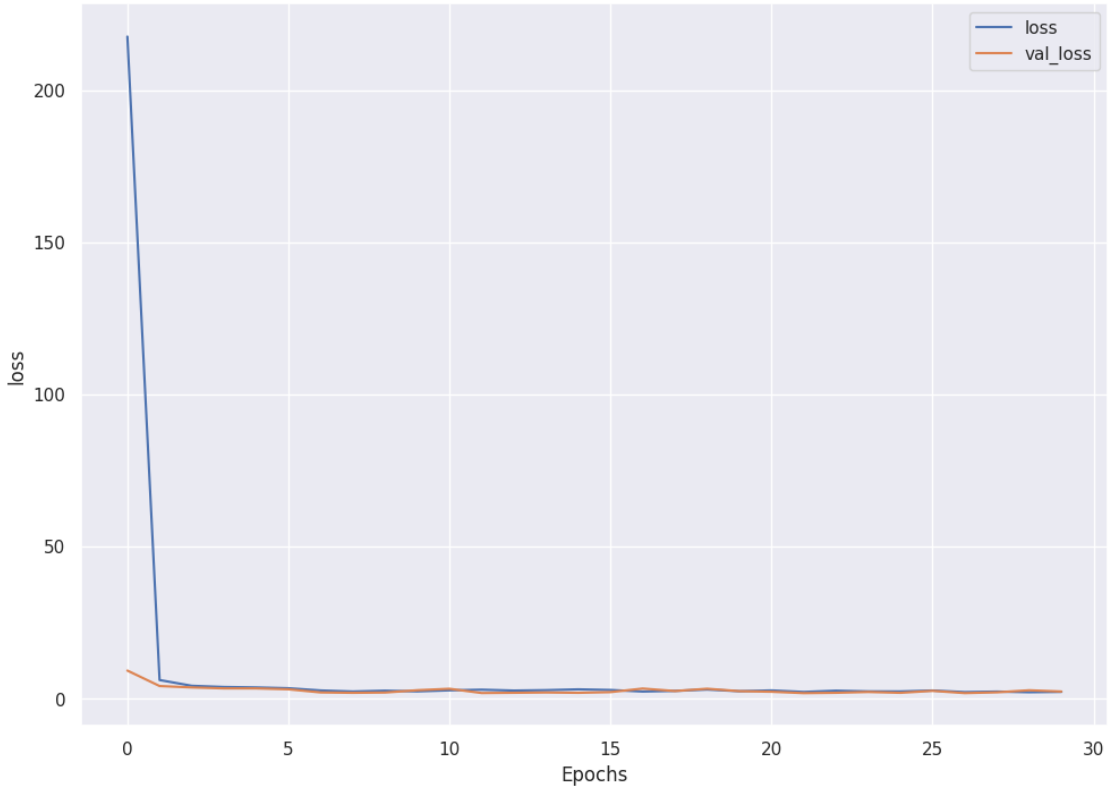
```
Epoch 1/30
219/219 [==============================] - 12s 46ms/step - loss: 217.6809 - mse:
217.6809 - mae: 9.1983 - val_loss: 9.1728 - val_mse: 9.1728 - val_mae: 2.2848
Epoch 2/30
219/219 [==============================] - 10s 47ms/step - loss: 6.0582 - mse:
6.0582 - mae: 1.8801 - val_loss: 4.0873 - val_mse: 4.0873 - val_mae: 1.5245
Epoch 3/30
219/219 [==============================] - 12s 55ms/step - loss: 4.1660 - mse:
4.1660 - mae: 1.5482 - val_loss: 3.6192 - val_mse: 3.6192 - val_mae: 1.4625
Epoch 4/30
219/219 [==============================] - 10s 47ms/step - loss: 3.7729 - mse:
3.7729 - mae: 1.4748 - val_loss: 3.3114 - val_mse: 3.3114 - val_mae: 1.3744
Epoch 5/30
219/219 [==============================] - 10s 47ms/step - loss: 3.6301 - mse:
3.6301 - mae: 1.4593 - val_loss: 3.3004 - val_mse: 3.3004 - val_mae: 1.3679
Epoch 6/30
219/219 [==============================] - 10s 44ms/step - loss: 3.3700 - mse:
3.3700 - mae: 1.4197 - val_loss: 3.0133 - val_mse: 3.0133 - val_mae: 1.3752
Epoch 7/30
219/219 [==============================] - 10s 47ms/step - loss: 2.6348 - mse:
2.6348 - mae: 1.2793 - val_loss: 1.9835 - val_mse: 1.9835 - val_mae: 1.1086
Epoch 8/30
219/219 [==============================] - 10s 47ms/step - loss: 2.2882 - mse:
2.2882 - mae: 1.1980 - val_loss: 1.8560 - val_mse: 1.8560 - val_mae: 1.0705
Epoch 9/30
219/219 [==============================] - 10s 44ms/step - loss: 2.5618 - mse:
2.5618 - mae: 1.2715 - val_loss: 1.9508 - val_mse: 1.9508 - val_mae: 1.1091
Epoch 10/30
219/219 [==============================] - 10s 45ms/step - loss: 2.3258 - mse:
2.3258 - mae: 1.2083 - val_loss: 2.7062 - val_mse: 2.7062 - val_mae: 1.3085
Epoch 11/30
219/219 [==============================] - 10s 47ms/step - loss: 2.6713 - mse:
2.6713 - mae: 1.2995 - val_loss: 3.1880 - val_mse: 3.1880 - val_mae: 1.4283
Epoch 12/30
219/219 [==============================] - 10s 47ms/step - loss: 2.8852 - mse:
2.8852 - mae: 1.3447 - val_loss: 1.7865 - val_mse: 1.7865 - val_mae: 1.0560
Epoch 13/30
219/219 [==============================] - 10s 46ms/step - loss: 2.5955 - mse:
```

```
2.5955 - mae: 1.2769 - val_loss: 1.8652 - val_mse: 1.8652 - val_mae: 1.0739
Epoch 14/30
219/219 [==============================] - 10s 45ms/step - loss: 2.7296 - mse:
2.7296 - mae: 1.3112 - val_loss: 1.9660 - val_mse: 1.9660 - val_mae: 1.1181
Epoch 15/30
219/219 [==============================] - 10s 47ms/step - loss: 2.9996 - mse:
2.9996 - mae: 1.3722 - val_loss: 1.8569 - val_mse: 1.8569 - val_mae: 1.0589
Epoch 16/30
219/219 [==============================] - 11s 50ms/step - loss: 2.7881 - mse:
2.7881 - mae: 1.3158 - val_loss: 2.0841 - val_mse: 2.0841 - val_mae: 1.1435
Epoch 17/30
219/219 [==============================] - 10s 46ms/step - loss: 2.2762 - mse:
2.2762 - mae: 1.1916 - val_loss: 3.2581 - val_mse: 3.2581 - val_mae: 1.4745
Epoch 18/30
219/219 [==============================] - 10s 44ms/step - loss: 2.4843 - mse:
2.4843 - mae: 1.2443 - val_loss: 2.4424 - val_mse: 2.4424 - val_mae: 1.2470
Epoch 19/30
219/219 [==============================] - 10s 47ms/step - loss: 2.9288 - mse:
2.9288 - mae: 1.3533 - val_loss: 3.1977 - val_mse: 3.1977 - val_mae: 1.4618
Epoch 20/30
219/219 [==============================] - 11s 50ms/step - loss: 2.3910 - mse:
2.3910 - mae: 1.2206 - val_loss: 2.4023 - val_mse: 2.4023 - val_mae: 1.2584
Epoch 21/30
219/219 [==============================] - 10s 47ms/step - loss: 2.6477 - mse:
2.6477 - mae: 1.2822 - val_loss: 2.2085 - val_mse: 2.2085 - val_mae: 1.1701
Epoch 22/30
219/219 [==============================] - 10s 46ms/step - loss: 2.1466 - mse:
2.1466 - mae: 1.1547 - val_loss: 1.7512 - val_mse: 1.7512 - val_mae: 1.0296
Epoch 23/30
219/219 [==============================] - 10s 47ms/step - loss: 2.5557 - mse:
2.5557 - mae: 1.2631 - val_loss: 1.8610 - val_mse: 1.8610 - val_mae: 1.0789
Epoch 24/30
219/219 [==============================] - 11s 49ms/step - loss: 2.3334 - mse:
2.3334 - mae: 1.2051 - val_loss: 2.1364 - val_mse: 2.1364 - val_mae: 1.1436
Epoch 25/30
219/219 [==============================] - 10s 48ms/step - loss: 2.3079 - mse:
2.3079 - mae: 1.2021 - val_loss: 1.8558 - val_mse: 1.8558 - val_mae: 1.0671
Epoch 26/30
219/219 [==============================] - 10s 45ms/step - loss: 2.6191 - mse:
2.6191 - mae: 1.2813 - val_loss: 2.5071 - val_mse: 2.5071 - val_mae: 1.2823
Epoch 27/30
219/219 [==============================] - 11s 48ms/step - loss: 2.1046 - mse:
2.1046 - mae: 1.1385 - val_loss: 1.7582 - val_mse: 1.7582 - val_mae: 1.0407
Epoch 28/30
219/219 [==============================] - 11s 50ms/step - loss: 2.2636 - mse:
2.2636 - mae: 1.1838 - val_loss: 2.0191 - val_mse: 2.0191 - val_mae: 1.1106
Epoch 29/30
219/219 [==============================] - 11s 48ms/step - loss: 2.0492 - mse:
```

```
2.0492 - mae: 1.1247 - val_loss: 2.6909 - val_mse: 2.6909 - val_mae: 1.3352
Epoch 30/30
219/219 [==============================] - 9s 43ms/step - loss: 2.1712 - mse:
2.1712 - mae: 1.1611 - val_loss: 2.2826 - val_mse: 2.2826 - val_mae: 1.2083
```

plotting the training history

[49]:
```
def plot_history(history,key):
    plt.plot(history.history[key])
    plt.plot(history.history['val_'+key])
    plt.xlabel("Epochs")
    plt.ylabel(key)
    plt.legend([key,'val_'+key])
    plt.show()
#plot the history
plot_history(history,'loss')
```



calculating performance metrics

[140]:
```
z_scaled = ANN_model.predict(X_test_ANN_scaled)
```

```
1094/1094 [==============================] - 6s 5ms/step
```

[141]:
```
r2_score(y_test_ANN_scaled, z_scaled)
```

[141]: 0.9812296597378213

[148]:
```
mse = mean_squared_error(y_test_ANN, z)
rmse = mse**.5
print("mse : ",mse)
print("rmse : ",rmse)
print("errors for neural net")
mae = mean_absolute_error(y_test_ANN, z)
print("mae : ",mae)
```

```
mse :  1.8401040600747824
rmse :  1.356504353135213
errors for neural net
mae :  1.0484327619750782
```

[149]:
```
mean_absolute_percentage_error(y_test_ANN, z)
```

[149]: 0.023490778239000434

scaling data before feeding to neural network and creating a custom TensorFlow layer that performs the scaling operation (alernate method)

[142]:
```
#␣
 ↪X_train_ANN,X_test_ANN,y_train_ANN,y_test_ANN=train_test_split(x,y,test_size=0.
 ↪2,random_state=42)
```

[143]:
```
# import tensorflow as tf
# from sklearn import preprocessing

# class MinMaxScalerLayer(tf.keras.layers.Layer):
#     def __init__(self, scaler_s, **kwargs):
#         super(MinMaxScalerLayer, self).__init__(**kwargs)
#         self.min_ = tf.Variable(initial_value=scaler_s.data_min_,␣
 ↪trainable=False, dtype=tf.float32)
#         self.scale_ = tf.Variable(initial_value=scaler_s.scale_,␣
 ↪trainable=False, dtype=tf.float32)

#     def call(self, inputs):
#         return (inputs - self.min_) * self.scale_
```

[144]:
```
# # Train-test split without scaling
# X_train_ANN, X_test_ANN, y_train_ANN, y_test_ANN = train_test_split(x, y,␣
 ↪test_size=0.2, random_state=42)

# # Initialize the MinMaxScaler and fit on the training data
# scaler_scaled = preprocessing.MinMaxScaler()
```

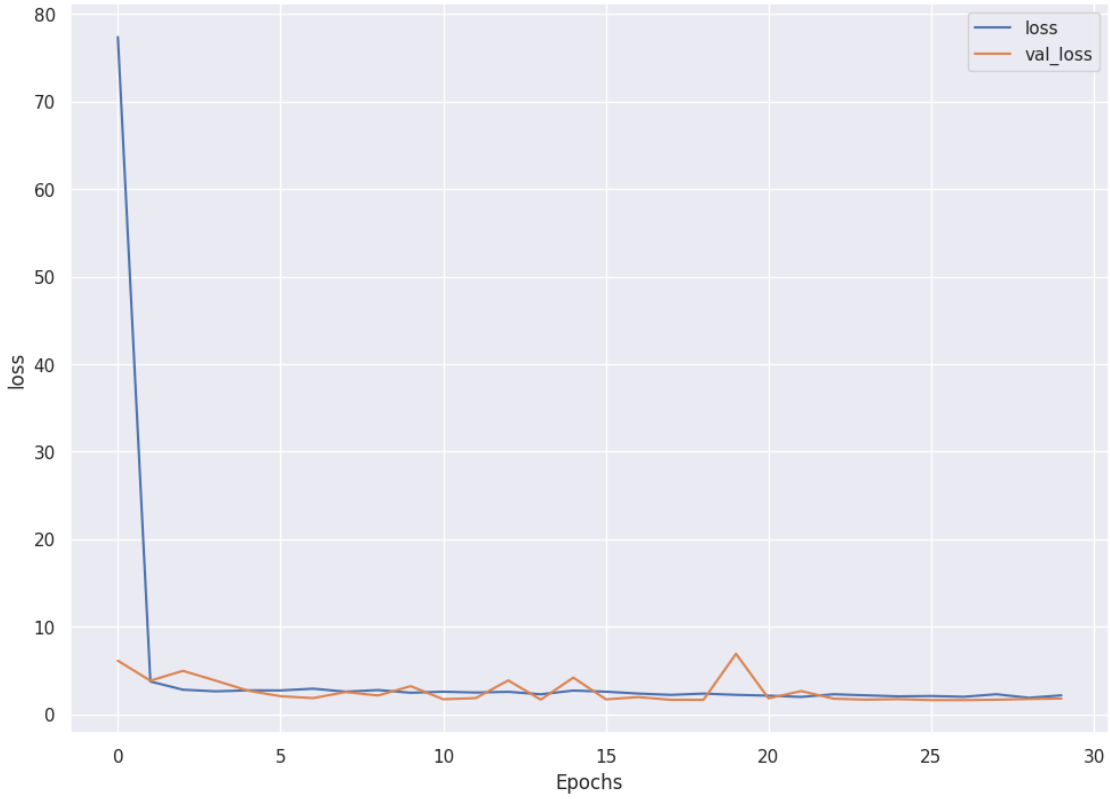
```
# scaler_scaled.fit(X_train_ANN)

# # Create a new sequential model
# ANN_model_scaled = tf.keras.Sequential()

# # Add the scaler layer to the model
# ANN_model_scaled.add(MinMaxScalerLayer(scaler_scaled,␣
 ↪input_shape=(X_train_ANN.shape[1],)))

# # Rest of your model architecture
# ANN_model_scaled.add(Dense(14, kernel_initializer='normal',␣
 ↪activation='relu'))
# ANN_model_scaled.add(Dense(512, activation='relu'))
# ANN_model_scaled.add(Dense(1024, activation='relu'))
# ANN_model_scaled.add(Dense(256, activation='relu'))
# ANN_model_scaled.add(Dense(1, activation='linear'))

# # Compile and train the model
# adam = Adam(learning_rate=0.01)
# ANN_model_scaled.compile(loss='mse', optimizer=adam, metrics=['mse', 'mae'])
# history = ANN_model_scaled.fit(X_train_ANN, y_train_ANN, epochs=30,␣
 ↪batch_size=512, verbose=1, validation_split=0.2)
```

```
Epoch 1/30
219/219 [==============================] - 19s 82ms/step - loss: 77.3646 - mse:
77.3646 - mae: 5.0478 - val_loss: 6.1211 - val_mse: 6.1211 - val_mae: 1.9568
Epoch 2/30
219/219 [==============================] - 17s 79ms/step - loss: 3.7414 - mse:
3.7414 - mae: 1.5002 - val_loss: 3.8366 - val_mse: 3.8366 - val_mae: 1.6054
Epoch 3/30
219/219 [==============================] - 18s 80ms/step - loss: 2.8101 - mse:
2.8101 - mae: 1.3280 - val_loss: 4.9620 - val_mse: 4.9620 - val_mae: 1.9026
Epoch 4/30
219/219 [==============================] - 17s 79ms/step - loss: 2.6207 - mse:
2.6207 - mae: 1.2849 - val_loss: 3.8538 - val_mse: 3.8538 - val_mae: 1.6082
Epoch 5/30
219/219 [==============================] - 18s 81ms/step - loss: 2.7313 - mse:
2.7313 - mae: 1.3157 - val_loss: 2.6951 - val_mse: 2.6951 - val_mae: 1.3363
Epoch 6/30
219/219 [==============================] - 18s 82ms/step - loss: 2.7227 - mse:
2.7227 - mae: 1.3083 - val_loss: 2.0646 - val_mse: 2.0646 - val_mae: 1.1295
Epoch 7/30
219/219 [==============================] - 17s 78ms/step - loss: 2.9241 - mse:
2.9241 - mae: 1.3666 - val_loss: 1.8462 - val_mse: 1.8462 - val_mae: 1.0708
Epoch 8/30
219/219 [==============================] - 17s 79ms/step - loss: 2.5818 - mse:
2.5818 - mae: 1.2747 - val_loss: 2.5377 - val_mse: 2.5377 - val_mae: 1.2885
```

```
Epoch 9/30
219/219 [==============================] - 18s 82ms/step - loss: 2.7673 - mse:
2.7673 - mae: 1.3195 - val_loss: 2.1464 - val_mse: 2.1464 - val_mae: 1.1560
Epoch 10/30
219/219 [==============================] - 18s 80ms/step - loss: 2.4600 - mse:
2.4600 - mae: 1.2420 - val_loss: 3.2136 - val_mse: 3.2136 - val_mae: 1.4381
Epoch 11/30
219/219 [==============================] - 17s 77ms/step - loss: 2.5795 - mse:
2.5795 - mae: 1.2662 - val_loss: 1.7182 - val_mse: 1.7182 - val_mae: 1.0252
Epoch 12/30
219/219 [==============================] - 18s 81ms/step - loss: 2.4844 - mse:
2.4844 - mae: 1.2454 - val_loss: 1.8483 - val_mse: 1.8483 - val_mae: 1.0699
Epoch 13/30
219/219 [==============================] - 17s 76ms/step - loss: 2.5660 - mse:
2.5660 - mae: 1.2667 - val_loss: 3.8670 - val_mse: 3.8670 - val_mae: 1.6364
Epoch 14/30
219/219 [==============================] - 18s 80ms/step - loss: 2.2841 - mse:
2.2841 - mae: 1.1859 - val_loss: 1.6720 - val_mse: 1.6720 - val_mae: 1.0112
Epoch 15/30
219/219 [==============================] - 18s 83ms/step - loss: 2.7089 - mse:
2.7089 - mae: 1.3020 - val_loss: 4.1910 - val_mse: 4.1910 - val_mae: 1.7212
Epoch 16/30
219/219 [==============================] - 18s 82ms/step - loss: 2.5793 - mse:
2.5793 - mae: 1.2701 - val_loss: 1.7064 - val_mse: 1.7064 - val_mae: 1.0179
Epoch 17/30
219/219 [==============================] - 18s 83ms/step - loss: 2.3666 - mse:
2.3666 - mae: 1.2127 - val_loss: 1.9628 - val_mse: 1.9628 - val_mae: 1.1205
Epoch 18/30
219/219 [==============================] - 19s 87ms/step - loss: 2.2150 - mse:
2.2150 - mae: 1.1738 - val_loss: 1.6524 - val_mse: 1.6524 - val_mae: 0.9973
Epoch 19/30
219/219 [==============================] - 19s 87ms/step - loss: 2.3610 - mse:
2.3610 - mae: 1.2032 - val_loss: 1.6513 - val_mse: 1.6513 - val_mae: 0.9987
Epoch 20/30
219/219 [==============================] - 19s 87ms/step - loss: 2.2194 - mse:
2.2194 - mae: 1.1714 - val_loss: 6.9084 - val_mse: 6.9084 - val_mae: 2.3159
Epoch 21/30
219/219 [==============================] - 18s 80ms/step - loss: 2.1262 - mse:
2.1262 - mae: 1.1443 - val_loss: 1.8176 - val_mse: 1.8176 - val_mae: 1.0637
Epoch 22/30
219/219 [==============================] - 18s 81ms/step - loss: 1.9833 - mse:
1.9833 - mae: 1.1047 - val_loss: 2.6490 - val_mse: 2.6490 - val_mae: 1.3082
Epoch 23/30
219/219 [==============================] - 17s 79ms/step - loss: 2.2862 - mse:
2.2862 - mae: 1.1944 - val_loss: 1.7825 - val_mse: 1.7825 - val_mae: 1.0515
Epoch 24/30
219/219 [==============================] - 17s 78ms/step - loss: 2.1635 - mse:
2.1635 - mae: 1.1579 - val_loss: 1.6716 - val_mse: 1.6716 - val_mae: 1.0095
```

```
Epoch 25/30
219/219 [==============================] - 17s 80ms/step - loss: 2.0474 - mse:
2.0474 - mae: 1.1278 - val_loss: 1.7312 - val_mse: 1.7312 - val_mae: 1.0296
Epoch 26/30
219/219 [==============================] - 18s 82ms/step - loss: 2.0860 - mse:
2.0860 - mae: 1.1397 - val_loss: 1.6237 - val_mse: 1.6237 - val_mae: 0.9887
Epoch 27/30
219/219 [==============================] - 17s 79ms/step - loss: 1.9998 - mse:
1.9998 - mae: 1.1122 - val_loss: 1.6185 - val_mse: 1.6185 - val_mae: 0.9900
Epoch 28/30
219/219 [==============================] - 18s 81ms/step - loss: 2.2873 - mse:
2.2873 - mae: 1.1865 - val_loss: 1.6676 - val_mse: 1.6676 - val_mae: 0.9992
Epoch 29/30
219/219 [==============================] - 18s 80ms/step - loss: 1.8838 - mse:
1.8838 - mae: 1.0777 - val_loss: 1.7432 - val_mse: 1.7432 - val_mae: 1.0459
Epoch 30/30
219/219 [==============================] - 17s 80ms/step - loss: 2.1644 - mse:
2.1644 - mae: 1.1580 - val_loss: 1.7986 - val_mse: 1.7986 - val_mae: 1.0381
```

[145]:
```
# def plot_history(history,key):
#     plt.plot(history.history[key])
#     plt.plot(history.history['val_'+key])
#     plt.xlabel("Epochs")
#     plt.ylabel(key)
#     plt.legend([key,'val_'+key])
#     plt.show()
# #plot the history
# plot_history(history,'loss')
```



[146]:
```
# z=ANN_model_scaled.predict(X_test_ANN)
```

```
1094/1094 [==============================] - 5s 5ms/step
```

[147]:
```
# r2_score(y_test_ANN, z)
```

[147]:
```
0.9789964570861258
```

0.2.4 Model Evaluation

saving the model

[150]:
```
joblib.dump(RF_regressor, 'RF.joblib')
```

[150]:
```
['RF.joblib']
```

[45]:
```
joblib.dump(scaler, "scaler.joblib")
```

[45]:
```
['scaler.joblib']
```

[46]:
```
# Saving the model in SavedModel format
ANN_model.save("ANN")
```

[155]:
```
# Saving the model in SavedModel format
# ANN_model_scaled.save("ANN_scaled")
```

loading the model saved before

[217]:
```
# Load the model
RF_loaded_model = joblib.load("RF.joblib")
```

[47]:
```
# Load the model from the SavedModel format
ANN_loaded_model = load_model("ANN")
# ANN_loaded_model_scaled = load_model("ANN_scaled")
#options = LoadOptions(experimental_io_device="/job:localhost")
#ANN_loaded_model = tf.keras.models.load_model("ANN", options=options)

# Load the MinMaxScaler using joblib
loaded_scaler = joblib.load("scaler.joblib")
```

testing the model

[48]:
```
ANN_loaded_model.predict(loaded_scaler.transform(x.iloc[0:1])).flatten()
```

```
1/1 [==============================] - 0s 71ms/step
```

[48]:
```
array([45.296677], dtype=float32)
```

[159]:
```
# ANN_loaded_model_scaled.predict(x.iloc[0:1]).flatten()
```

```
1/1 [==============================] - 0s 123ms/step
```

[159]:
```
array([46.29598], dtype=float32)
```

defining predictor functions

[230]:
```
def RF_regressor_predictor(sample):

    """
    Predict the target value for a single sample using a regression model.
    """

    # Extract and return the predicted value
    return RF_loaded_model.predict(sample)
```

[206]:
```
def RF_regressor_predictor_2(sample):

    """
    Predict the target value for a single sample using a regression model.
    """

    # Extract and return the predicted value
    return RF_regressor.predict(sample)
```

[105]:
```
def ANN_regressor_predictor(sample):

    """
    Predict the target value for a single sample using a regression model.
    """

    # Extract and return the predicted value
    return ANN_loaded_model.predict(loaded_scaler.transform(sample)).flatten()
```

[104]:
```
def ANN_regressor_predictor_2(sample):

    """
    Predict the target value for a single sample using a regression model.
    """

    # Extract and return the predicted value
    return ANN_model.predict(loaded_scaler.transform(sample)).flatten()
```

[163]:
```
def ANN_regressor_predictor_scaled(sample):

    """
    Predict the target value for a single sample using a regression model.
    """

    # Extract and return the predicted value
    return ANN_loaded_model_scaled.predict(sample).flatten()
```

testing the predictor functions

[52]:
```
example_input = x.iloc[0:1]
```

[220]:
```
RF_regressor_predictor(example_input)
```

[220]:
```
array([46.39])
```

[208]:
```
RF_regressor_predictor_2(example_input)
```

[208]:
```
array([46.39])
```

[106]:
```
ANN_regressor_predictor(example_input)
```

```
1/1 [==============================] - 0s 17ms/step
```

[106]:
```
array([45.296677], dtype=float32)
```

[107]:
```
ANN_regressor_predictor_2(example_input)
```

```
1/1 [==============================] - 0s 20ms/step
```

[107]:
```
array([45.296677], dtype=float32)
```

[169]:
```
ANN_regressor_predictor_scaled(example_input)
```

```
1/1 [==============================] - 0s 34ms/step
```

[169]:
```
array([46.29598], dtype=float32)
```

0.2.5 Model deployment

deploying API

[62]:
```
mb = modelbit.login()
```

```
<IPython.core.display.HTML object>
```

[231]:
```
mb.deploy(RF_regressor_predictor,
          dataframe_mode=True,
          example_dataframe=example_input, extra_files = "RF.joblib",␣
 ↪python_packages=["scikit-learn==1.2.2"])
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

[ ]:
```
mb.deploy(RF_regressor_predictor_2,
          dataframe_mode=True,
          example_dataframe=example_input, python_packages=["scikit-learn==1.2.
 ↪2"])
```

[108]:
```
mb.deploy(ANN_regressor_predictor,
          dataframe_mode=True,
          example_dataframe=example_input, python_packages=["tensorflow==2.14.
 ↪0"])
```

```
<IPython.core.display.HTML object>
```

```
Uploading 'ANN_loaded_model': 100%|      | 2.93M/2.93M [00:00<00:00,
4.05MB/s]
```

```
<IPython.core.display.HTML object>
```

[109]:
```
mb.deploy(ANN_regressor_predictor_2,
          dataframe_mode=True,
          example_dataframe=example_input, python_packages=["tensorflow==2.14.
 ↪0", "numpy==1.23.5"])
```

```
<IPython.core.display.HTML object>
```

```
Uploading 'ANN_model': 100%|      | 7.79M/7.79M [00:00<00:00, 9.31MB/s]
```

```
<IPython.core.display.HTML object>
```

testing the API

[232]:
```
# API on modelbit
import requests, json

requests.post(
    "https://vishwarathtomar.app.modelbit.com/v1/RF_regressor_predictor/latest",
    headers={"Content-Type":"application/json"},
    data=json.dumps({"data": {
        "store_id": 0,
        "total_items": 4,
        "distinct_items": 4,
        "subtotal": 39.1,
        "total_onshift_runners": 33,
        "total_busy_runners": 14,
        "total_outstanding_orders":21,
        "store_to_consumer_driving_duration_mins": 14,
        "created_at_hour": 17,
        "created_at_day": 4
    }})).json()
```

[232]:
```
{'data': 46.39}
```

[216]:
```
# API on modelbit
import requests, json

requests.post(
    "https://vishwarathtomar.app.modelbit.com/v1/RF_regressor_predictor_2/
 ↪latest",
    headers={"Content-Type":"application/json"},
    data=json.dumps({"data": {
        "store_id": 0,
        "total_items": 4,
        "distinct_items": 4,
        "subtotal": 39.1,
        "total_onshift_runners": 33,
        "total_busy_runners": 14,
        "total_outstanding_orders":21,
        "store_to_consumer_driving_duration_mins": 14,
        "created_at_hour": 17,
        "created_at_day": 4
    }})).json()
```

[216]:
```
{'data': 46.39}
```

[111]:
```
# API on modelbit
import requests, json
```

```
requests.post(
    "https://vishwarathtomar.app.modelbit.com/v1/ANN_regressor_predictor/
 ↪latest",
    headers={"Content-Type":"application/json"},
    data=json.dumps({"data": {
        "store_id": 0,
        "total_items": 4,
        "distinct_items": 4,
        "subtotal": 39.1,
        "total_onshift_runners": 33,
        "total_busy_runners": 14,
        "total_outstanding_orders":21,
        "store_to_consumer_driving_duration_mins": 14,
        "created_at_hour": 17,
        "created_at_day": 4
    }})).json()
```

[111]:
```
{'data': 45.29667663574219}
```

[110]:
```
# API on modelbit
import requests, json

requests.post(
    "https://vishwarathtomar.app.modelbit.com/v1/ANN_regressor_predictor_2/
 ↪latest",
    headers={"Content-Type":"application/json"},
    data=json.dumps({"data": {
        "store_id": 0,
        "total_items": 4,
        "distinct_items": 4,
        "subtotal": 39.1,
        "total_onshift_runners": 33,
        "total_busy_runners": 14,
        "total_outstanding_orders":21,
        "store_to_consumer_driving_duration_mins": 14,
        "created_at_hour": 17,
        "created_at_day": 4
    }})).json()
```

[110]:
```
{'data': 45.29667663574219}
```

[ ]:
```
# API on local host
import requests, json

requests.post(
    "http://localhost:8605/v1/models/my_model:predict",
    headers={"Content-Type":"application/json"},
```

```
    data=json.dumps({"data": {
        "store_id": 0,
        "total_items": 4,
        "distinct_items": 4,
        "subtotal": 39.1,
        "total_onshift_runners": 33,
        "total_busy_runners": 14,
        "total_outstanding_orders":21,
        "store_to_consumer_driving_duration_mins": 14,
        "created_at_hour": 17,
        "created_at_day": 4
    }})).json()
```

[1]:
```
!export PATH=/Library/TeX/texbin:$PATH
```

```
Der Befehl "export" ist entweder falsch geschrieben oder
konnte nicht gefunden werden.
```

[ ]:
```

```