rter-neural-networks-business-case

September 1, 2024

[ ]:
```
import numpy as np
import pandas as pd
```

[ ]:
```
df = pd.read_csv('porter_dataset.csv')
```

[ ]:
```
# Problem statement has been defined below
```

1 Problem Statement

The dataset contains details of Porter Company, which is India's Largest Marketplace for Intra-City Logistics.

Porter works with a wide range of restaurants for delivering their items directly to the people.

Porter has a number of delivery partners available for delivering the food, from various restaurants and wants to get an estimated delivery time that it can provide the customers on the basis of what they are ordering, from where and also the delivery partners.

This dataset has the required data to train a regression model that will do the delivery time estimation, based on all those features

[ ]:
```
# Data pre processing
```

[ ]:
```
df.shape
# we have 1,97,428 rows and 14 columns
```

[ ]:
```
(197428, 14)
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 197428 entries, 0 to 197427
Data columns (total 14 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   market_id                 196441 non-null  float64
 1   created_at                197428 non-null  object
 2   actual_delivery_time      197421 non-null  object
 3   store_id                  197428 non-null  object
```

```
 4   store_primary_category    192668 non-null  object
 5   order_protocol            196433 non-null  float64
 6   total_items               197428 non-null  int64
 7   subtotal                  197428 non-null  int64
 8   num_distinct_items        197428 non-null  int64
 9   min_item_price            197428 non-null  int64
 10  max_item_price            197428 non-null  int64
 11  total_onshift_partners    181166 non-null  float64
 12  total_busy_partners       181166 non-null  float64
 13  total_outstanding_orders  181166 non-null  float64
dtypes: float64(5), int64(5), object(4)
memory usage: 21.1+ MB
```

[ ]:
```
df.describe()
```

[ ]:
```
           market_id  order_protocol    total_items       subtotal  \
count  196441.000000   196433.000000  197428.000000  197428.000000
mean        2.978706        2.882352       3.196391    2682.331402
std         1.524867        1.503771       2.666546    1823.093688
min         1.000000        1.000000       1.000000       0.000000
25%         2.000000        1.000000       2.000000    1400.000000
50%         3.000000        3.000000       3.000000    2200.000000
75%         4.000000        4.000000       4.000000    3395.000000
max         6.000000        7.000000     411.000000   27100.000000

       num_distinct_items  min_item_price  max_item_price  \
count       197428.000000   197428.000000   197428.000000
mean             2.670791      686.218470     1159.588630
std              1.630255      522.038648      558.411377
min              1.000000      -86.000000        0.000000
25%              1.000000      299.000000      800.000000
50%              2.000000      595.000000     1095.000000
75%              3.000000      949.000000     1395.000000
max             20.000000    14700.000000    14700.000000

       total_onshift_partners  total_busy_partners  total_outstanding_orders
count           181166.000000        181166.000000             181166.000000
mean                44.808093            41.739747                 58.050065
std                 34.526783            32.145733                 52.661830
min                 -4.000000            -5.000000                 -6.000000
25%                 17.000000            15.000000                 17.000000
50%                 37.000000            34.000000                 41.000000
75%                 65.000000            62.000000                 85.000000
max                171.000000           154.000000                285.000000
```

[ ]:
```
df.columns
```

[ ]:
```
Index(['market_id', 'created_at', 'actual_delivery_time', 'store_id',
       'store_primary_category', 'order_protocol', 'total_items', 'subtotal',
       'num_distinct_items', 'min_item_price', 'max_item_price',
       'total_onshift_partners', 'total_busy_partners',
       'total_outstanding_orders'],
      dtype='object')
```

[ ]:
```
df.duplicated().sum()
# we do not have any duplicated rows
```

[ ]:
```
0
```

[ ]:
```
df.isnull().sum()
# yes we have some null values
```

[ ]:
```
market_id                   987
created_at                    0
actual_delivery_time          7
store_id                      0
store_primary_category     4760
order_protocol              995
total_items                   0
subtotal                      0
num_distinct_items            0
min_item_price                0
max_item_price                0
total_onshift_partners    16262
total_busy_partners       16262
total_outstanding_orders  16262
dtype: int64
```

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

[ ]:
```
   market_id           created_at actual_delivery_time  \
0        1.0  2015-02-06 22:24:17  2015-02-06 23:27:16
1        2.0  2015-02-10 21:49:25  2015-02-10 22:56:29
2        3.0  2015-01-22 20:39:28  2015-01-22 21:09:09
3        3.0  2015-02-03 21:21:45  2015-02-03 22:13:00
4        3.0  2015-02-15 02:40:36  2015-02-15 03:20:26

                          store_id store_primary_category  order_protocol  \
0  df263d996281d984952c07998dc54358               american             1.0
1  f0ade77b43923b38237db569b016ba25                mexican             2.0
2  f0ade77b43923b38237db569b016ba25                    NaN             1.0
3  f0ade77b43923b38237db569b016ba25                    NaN             1.0
4  f0ade77b43923b38237db569b016ba25                    NaN             1.0
```

```
   total_items  subtotal  num_distinct_items  min_item_price  max_item_price  \
0            4      3441                   4             557            1239
1            1      1900                   1            1400            1400
2            1      1900                   1            1900            1900
3            6      6900                   5             600            1800
4            3      3900                   3            1100            1600

   total_onshift_partners  total_busy_partners  total_outstanding_orders
0                    33.0                 14.0                      21.0
1                     1.0                  2.0                       2.0
2                     1.0                  0.0                       0.0
3                     1.0                  1.0                       2.0
4                     6.0                  6.0                       9.0
```

[ ]:
```
df['market_id'].value_counts(normalize= True).sort_values() * 100
# Since most orders are from market_id '2.0' we fill the NA values in market_id␣
↪column with '2.0'
```

[ ]:
```
market_id
6.0     7.355898
5.0     9.163057
3.0    11.859541
1.0    19.363066
4.0    24.230685
2.0    28.027754
Name: proportion, dtype: float64
```

[ ]:
```
df['market_id'].fillna(2.0, inplace= True)
df['market_id'].isnull().sum()
# No NA values are present in the market_id column
```

[ ]:
```
0
```

[ ]:
```
df['store_primary_category'].value_counts(normalize= True).sort_values() * 100
```

[ ]:
```
store_primary_category
alcohol-plus-food     0.000519
chocolate             0.000519
belgian               0.001038
indonesian            0.001038
lebanese              0.004671
                         …
sandwich              5.221417
burger                5.687504
mexican               8.874852
pizza                 8.990076
american             10.068615
```

```
Name: proportion, Length: 74, dtype: float64
```

[ ]:
```
# filling the NA values in store_primary_category column with 'other'
df['store_primary_category'].fillna('other', inplace= True)
df['store_primary_category'].isnull().sum()
```

[ ]:
```
0
```

[ ]:
```
df['order_protocol'].value_counts(normalize= True).sort_values() * 100
```

[ ]:
```
order_protocol
7.0     0.009673
6.0     0.404209
4.0     9.852723
2.0    12.244378
5.0    22.547128
3.0    27.082517
1.0    27.859372
Name: proportion, dtype: float64
```

[ ]:
```
# replacing 'order_protocol' column's na values with 1.0
df['order_protocol'].fillna(1.0, inplace= True)
df['order_protocol'].isnull().sum()
```

[ ]:
```
0
```

[ ]:
```
df['actual_delivery_time'].value_counts()
```

[ ]:
```
actual_delivery_time
2015-02-11 20:40:45    5
2015-02-16 03:51:49    5
2015-02-12 03:14:14    5
2015-01-24 03:41:03    5
2015-02-01 03:44:13    5
                      ..
2015-02-10 02:42:43    1
2015-02-14 04:07:50    1
2015-02-15 22:06:14    1
2015-02-17 04:24:59    1
2015-02-08 20:01:41    1
Name: count, Length: 178110, dtype: int64
```

[ ]:
```
df['actual_delivery_time'].mode()[0]
```

[ ]:
```
'2015-01-24 03:41:03'
```

[ ]:
```
# replacing na values in 'actual_delivery_time'
# the 'actual_delivery_time' is a timestamp and so I prefer to replace the na␣
 ↪values with mode of this column
df['actual_delivery_time'] = df['actual_delivery_time'].astype(str)
df['actual_delivery_time'].fillna(df['actual_delivery_time'].mode()[0],␣
 ↪inplace= True)
df['actual_delivery_time'].isnull().sum()
```

[ ]: 0

[ ]:
```
# replacing the na values in 'total_onshift_partners', 'total_busy_partners',␣
 ↪'total_outstanding_orders' with their respective averages

ls = ['total_onshift_partners', 'total_busy_partners',␣
 ↪'total_outstanding_orders']
for i in ls:
    df[i].fillna(int(df[i].mean()), inplace= True)
    print(f"null values in {i}", df[i].isna().sum())
```

```
null values in total_onshift_partners 0
null values in total_busy_partners 0
null values in total_outstanding_orders 0
```

[ ]:
```
df.isna().sum()
# we've replaced all na values.
# The dataset has no na values now
```

[ ]:
```
market_id                 0
created_at                0
actual_delivery_time      0
store_id                  0
store_primary_category    0
order_protocol            0
total_items               0
subtotal                  0
num_distinct_items        0
min_item_price            0
max_item_price            0
total_onshift_partners    0
total_busy_partners       0
total_outstanding_orders  0
dtype: int64
```

[ ]:
```
df.duplicated().sum()
# the dataset has no duplicated rows
```

[ ]: 0

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 197428 entries, 0 to 197427
Data columns (total 14 columns):
 #   Column                   Non-Null Count   Dtype
---  ------                   --------------   -----
 0   market_id                197428 non-null  float64
 1   created_at               197428 non-null  object
 2   actual_delivery_time     197428 non-null  object
 3   store_id                 197428 non-null  object
 4   store_primary_category   197428 non-null  object
 5   order_protocol           197428 non-null  float64
 6   total_items              197428 non-null  int64
 7   subtotal                 197428 non-null  int64
 8   num_distinct_items       197428 non-null  int64
 9   min_item_price           197428 non-null  int64
 10  max_item_price           197428 non-null  int64
 11  total_onshift_partners   197428 non-null  float64
 12  total_busy_partners      197428 non-null  float64
 13  total_outstanding_orders 197428 non-null  float64
dtypes: float64(5), int64(5), object(4)
memory usage: 21.1+ MB
```

[ ]:
```
(df['total_onshift_partners'] - df['total_onshift_partners'].astype(int)).
 ↪value_counts()
for i in ls:
    print("number of values which have only '0' after the decimal point",(df[i]␣
 ↪- df[i].astype(int)).value_counts())
```

```
number of values which have only '0' after the decimal point
total_onshift_partners
0.0    197428
Name: count, dtype: int64
number of values which have only '0' after the decimal point total_busy_partners
0.0    197428
Name: count, dtype: int64
number of values which have only '0' after the decimal point
total_outstanding_orders
0.0    197428
Name: count, dtype: int64
```

[ ]:
```
# Based on df.info() we change the datatypes as follow
# 'market_id', 'order_protocol'  to object
# 'total_onshift_partners', 'total_busy_partners', 'total_outstanding_orders'␣
 ↪to int64
# 'created_at', 'actual_delivery_time' to timestamp
```

```
# 'min_item_price', 'max_item_price' to float64
```

[ ]:
```
for i in ['market_id', 'order_protocol']:
  df[i] = df[i].astype('object')
for i in ['total_onshift_partners', 'total_busy_partners',␣
 ↪'total_outstanding_orders']:
  df[i] = df[i].astype('int64')
for i in ['created_at', 'actual_delivery_time']:
  df[i] = pd.to_datetime(df[i])
for i in ['min_item_price', 'max_item_price']:
  df[i] = df[i].astype('float64')
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 197428 entries, 0 to 197427
Data columns (total 14 columns):
 #   Column                   Non-Null Count   Dtype
---  ------                   --------------   -----
 0   market_id                197428 non-null  object
 1   created_at               197428 non-null  datetime64[ns]
 2   actual_delivery_time     197421 non-null  datetime64[ns]
 3   store_id                 197428 non-null  object
 4   store_primary_category   197428 non-null  object
 5   order_protocol           197428 non-null  object
 6   total_items              197428 non-null  int64
 7   subtotal                 197428 non-null  int64
 8   num_distinct_items       197428 non-null  int64
 9   min_item_price           197428 non-null  float64
 10  max_item_price           197428 non-null  float64
 11  total_onshift_partners   197428 non-null  int64
 12  total_busy_partners      197428 non-null  int64
 13  total_outstanding_orders 197428 non-null  int64
dtypes: datetime64[ns](2), float64(2), int64(6), object(4)
memory usage: 21.1+ MB
```

[ ]:
```
# Convert the following columns to category format:
# 'market_id', 'store_id', 'store_primary_category', 'order_protocol'

df["market_id"] = df["market_id"].astype("category")
df["store_id"] = df["store_id"].astype("category")
df["store_primary_category"] = df["store_primary_category"].astype("category")
df["order_protocol"] = df["order_protocol"].astype("category")
```

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

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 197428 entries, 0 to 197427
Data columns (total 14 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   market_id                 197428 non-null  category
 1   created_at                197428 non-null  datetime64[ns]
 2   actual_delivery_time      197421 non-null  datetime64[ns]
 3   store_id                  197428 non-null  category
 4   store_primary_category    197428 non-null  category
 5   order_protocol            197428 non-null  category
 6   total_items               197428 non-null  int64
 7   subtotal                  197428 non-null  int64
 8   num_distinct_items        197428 non-null  int64
 9   min_item_price            197428 non-null  float64
 10  max_item_price            197428 non-null  float64
 11  total_onshift_partners    197428 non-null  int64
 12  total_busy_partners       197428 non-null  int64
 13  total_outstanding_orders  197428 non-null  int64
dtypes: category(4), datetime64[ns](2), float64(2), int64(6)
memory usage: 16.3 MB
```

[ ]:
```
# extracting the hour and day from created_at & actual delivery_time columns
df['hour']=df['created_at'].dt.hour
df['day']=df['created_at'].dt.dayofweek
```

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

[ ]:

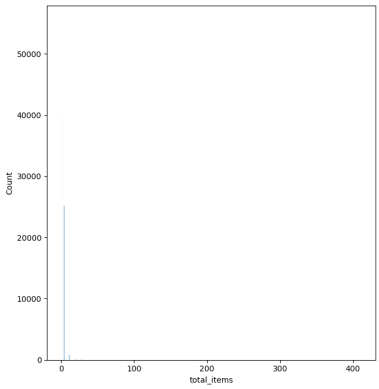
	market_id	created_at	actual_delivery_time	store_id	store_primary_category	order_protocol	total_items	subtotal	num_distinct_items	min_item_price	max_item_price
0	1.0	2015-02-06 22:24:17	2015-02-06 23:27:16	df263d996281d984952c07998dc54358	american	1.0	4	3441	4	557.0	1239.0
1	2.0	2015-02-10 21:49:25	2015-02-10 22:56:29	f0ade77b43923b38237db569b016ba25	mexican	2.0	1	1900	1	1400.0	1400.0
2	3.0	2015-01-22 20:39:28	2015-01-22 21:09:09	f0ade77b43923b38237db569b016ba25	other	1.0	1	1900	1	1900.0	1900.0
3	3.0	2015-02-03 21:21:45	2015-02-03 22:13:00	f0ade77b43923b38237db569b016ba25	other	1.0	6	6900	5	600.0	1800.0
4	3.0	2015-02-15 02:40:36	2015-02-15 03:20:26	f0ade77b43923b38237db569b016ba25	other	1.0	3	3900	3	1100.0	1600.0

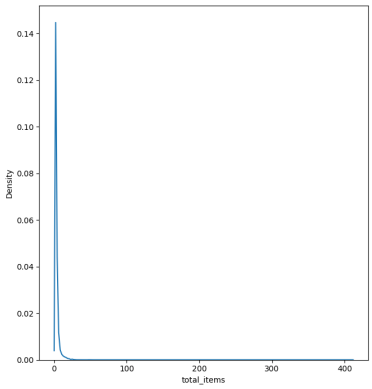
```
   total_onshift_partners  total_busy_partners  total_outstanding_orders  \
0                      33                   14                        21
1                       1                    2                         2
2                       1                    0                         0
3                       1                    1                         2
4                       6                    6                         9

   hour  day
0    22    4
1    21    1
2    20    3
3    21    1
4     2    6
```

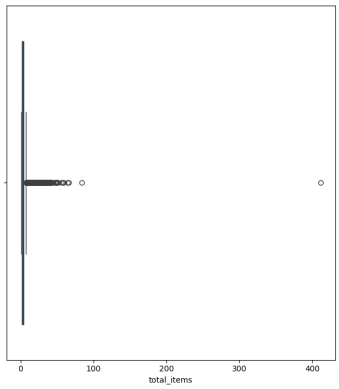
[ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns

def hist_kde_box_plot(df, var:str):
    fig,axes = plt.subplots(nrows=1, ncols=3, figsize=(20,7))
    sns.histplot(df, x = var, ax=axes[0])
    sns.kdeplot(df, x = var, ax=axes[1])
    sns.boxplot(df, x = var, ax=axes[2])
    plt.tight_layout()
    plt.show()
```

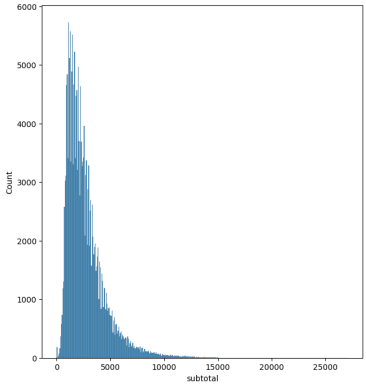
[ ]:
```
# Univariate Analysis
```

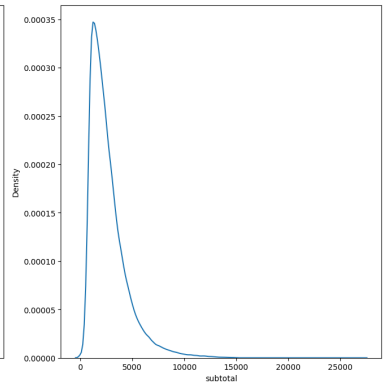
[ ]:
```
ls =␣
 ↪['total_items','subtotal','num_distinct_items','min_item_price','max_item_price','total_onsi

for i in ls:
    hist_kde_box_plot(df,i)


# From box plots we can see there are outliers for all the columns in the above␣
 ↪list 'ls'
```

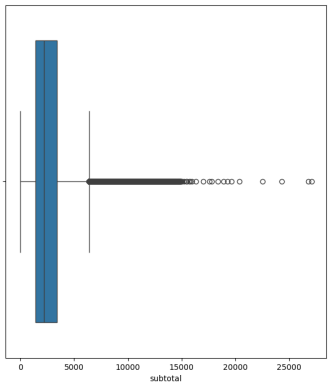

Count
total_items

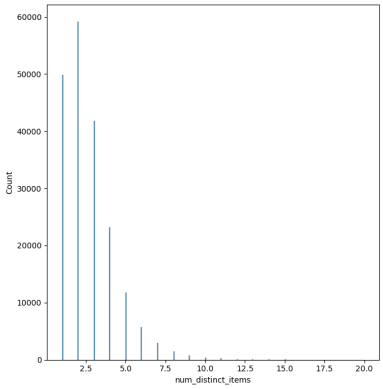

Density
total_items

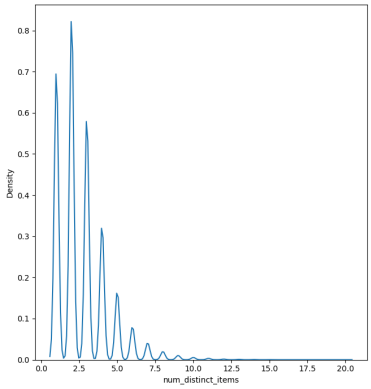

total_items

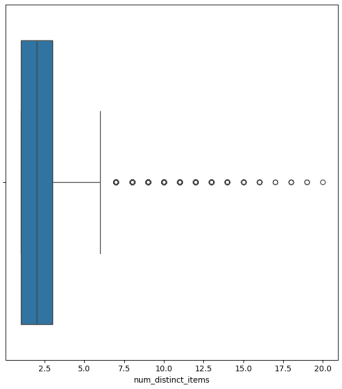

Count
subtotal

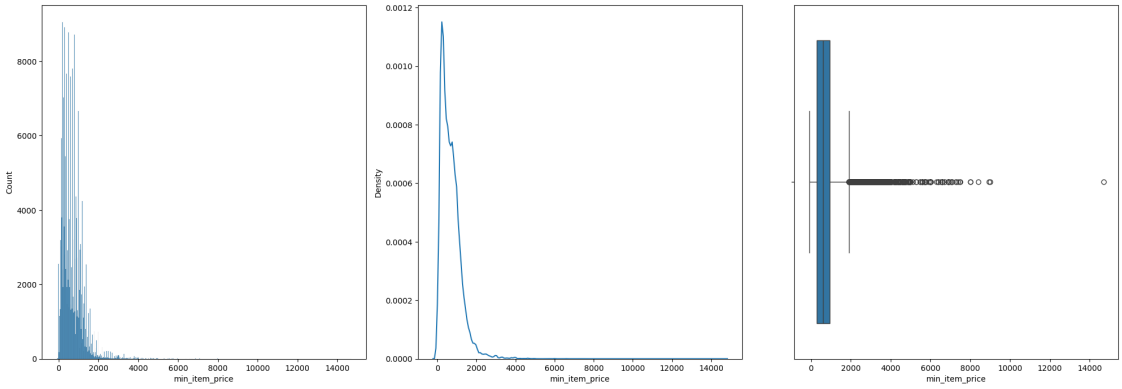

Density
subtotal

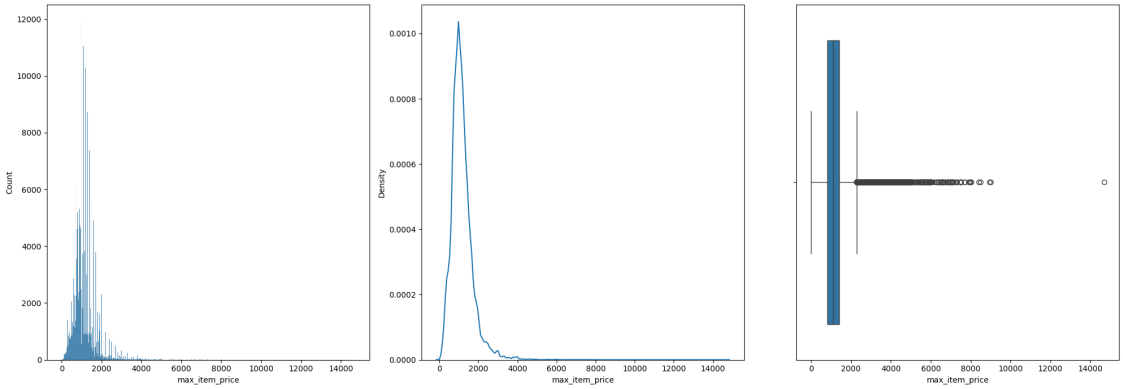

subtotal

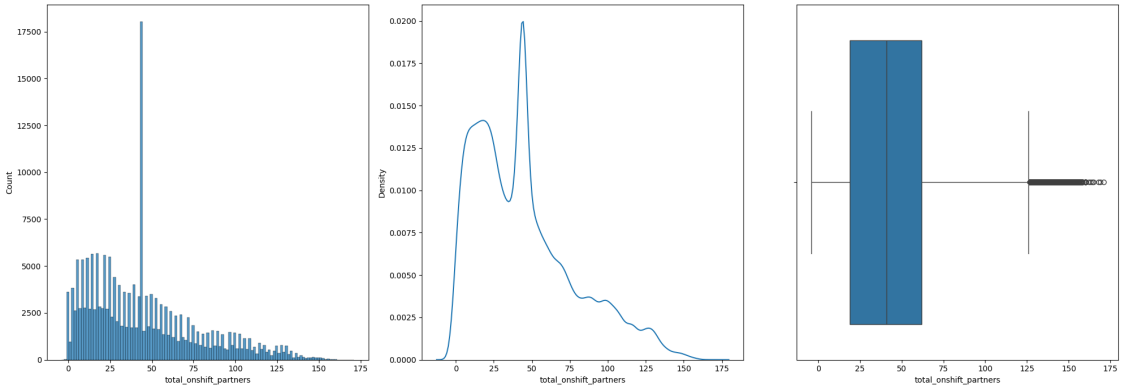

Count
num_distinct_items

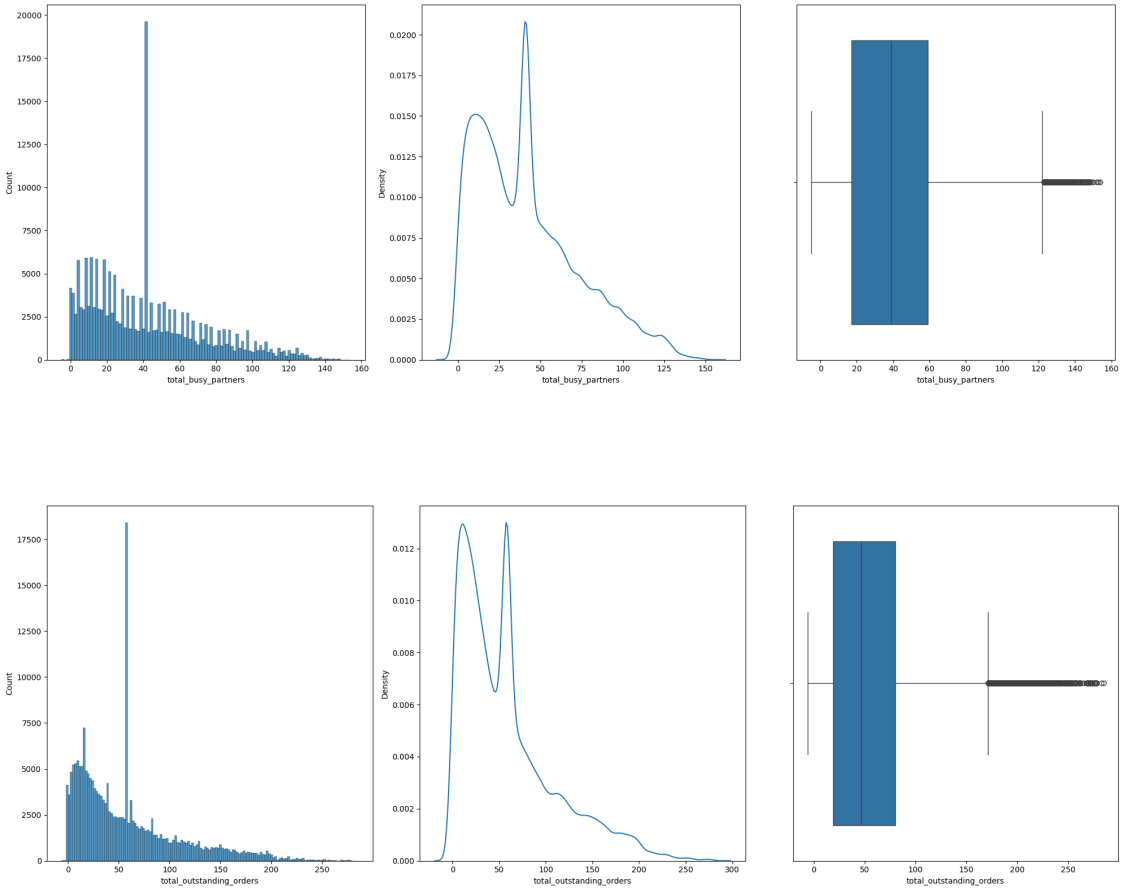

Density
num_distinct_items


num_distinct_items


Count
min_item_price
Density
min_item_price
min_item_price

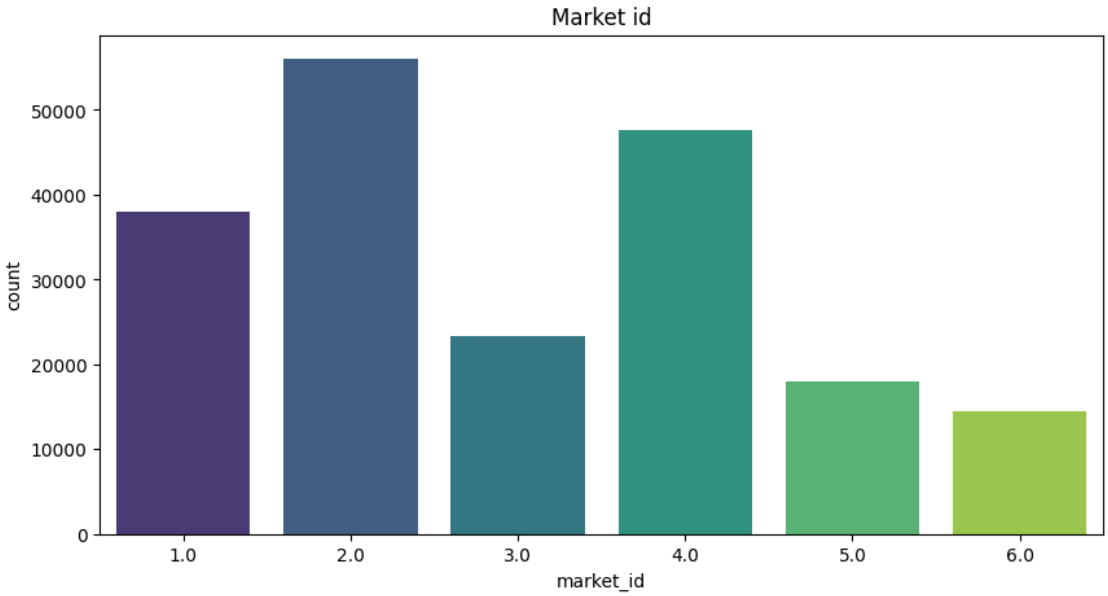

Count
max_item_price
Density
max_item_price
max_item_price


Count
total_onshift_partners
Density
total_onshift_partners
total_onshift_partners



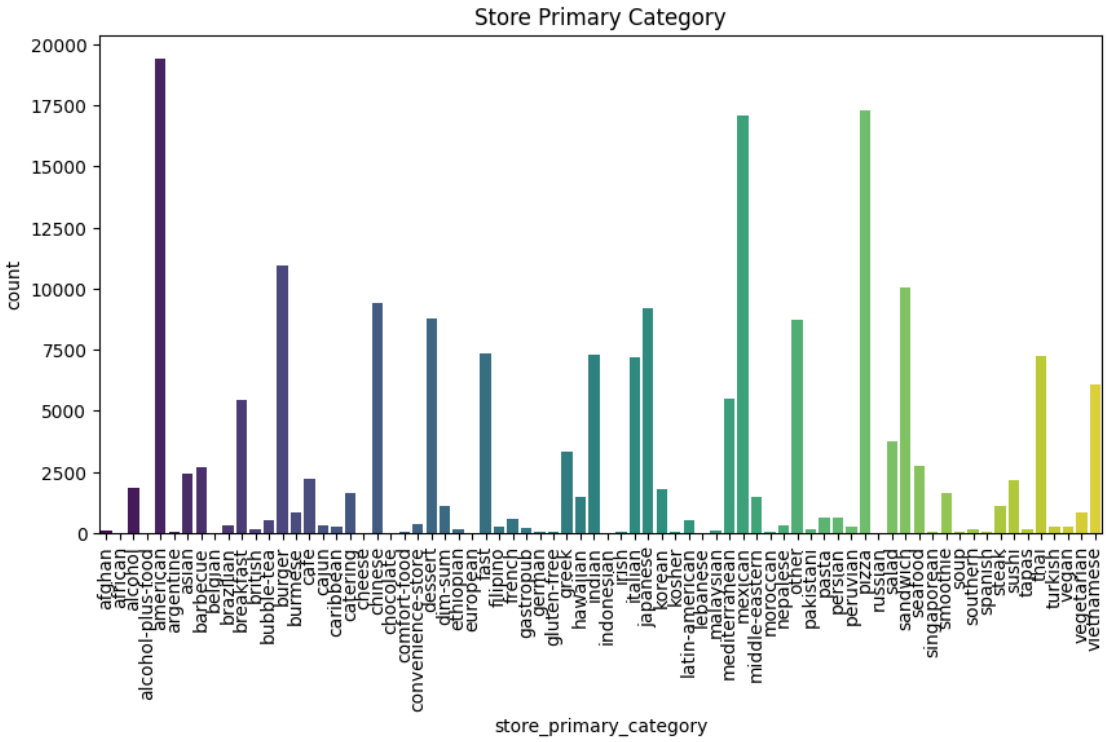
[ ]:
```
fig,ax = plt.subplots(figsize=(10,5))
sns.countplot(df, x="market_id",ax=ax,palette="viridis")
ax.set_title("Market id")
plt.show()

# we can see market_id has highest number of orders
```

```
<ipython-input-912-f18cc2570710>:2: FutureWarning:


Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.
```



[ ]:
```
fig,ax = plt.subplots(figsize=(10,5))
sns.countplot(df, x="store_primary_category",ax=ax, palette='viridis')
ax.set_title("Store Primary Category")
ax.set_xticklabels(ax.get_xticklabels(), rotation=90)
plt.show()

# American style restaurants are more in the dataset
# followed by Mexican and restaurants/outlets selling pizza
```

```
<ipython-input-913-71deb258dbf7>:2: FutureWarning:


Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.


<ipython-input-913-71deb258dbf7>:4: UserWarning:

FixedFormatter should only be used together with FixedLocator
```



[ ]:
```
import plotly.express as px

hour_day_counts = df.groupby(['hour', 'day']).size().reset_index(name='count')

# Plotting using Plotly
fig = px.scatter(hour_day_counts, x='hour', y='count', color='day',␣
 ↪hover_data=['hour', 'count', 'day'])
fig.update_layout(
    title="Counts by Hour and Day",
    xaxis=dict(tickmode='linear', tick0=0, dtick=1),  # Set x-axis ticks for␣
 ↪each hour
    title_x=0.5
)
fig.show()

# we can see the time trend at wich orders are being placed
# clearly on day 6 (Saturday) more order are being placed at all points of time
# more order are placed at hour 1,2,3 i.e., 1AM,2AM,3AM
```

[ ]:
```
df['total_items'].mean()
# average number of items = 3.19
```

[ ]:
```
3.196390582896043
```

[ ]:
```
df['subtotal'].mean()
# average spend per order = 2682.33
```

[ ]:
```
2682.331401827502
```

[ ]:
```
df.describe()
```

[ ]:
```
                          created_at           actual_delivery_time  \
count                         197428                         197421
mean   2015-02-04 22:00:09.537962752  2015-02-04 22:48:23.348914432
min              2014-10-19 05:24:15            2015-01-21 15:58:11
25%              2015-01-29 02:32:42            2015-01-29 03:22:29
50%       2015-02-05 03:29:09.500000            2015-02-05 04:40:41
75%       2015-02-12 01:39:18.500000            2015-02-12 02:25:26
max              2015-02-18 06:00:44            2015-02-19 22:45:31
std                              NaN                            NaN

         total_items       subtotal  num_distinct_items  min_item_price  \
count  197428.000000  197428.000000       197428.000000   197428.000000
mean        3.196391    2682.331402            2.670791      686.218470
min         1.000000       0.000000            1.000000      -86.000000
25%         2.000000    1400.000000            1.000000      299.000000
50%         3.000000    2200.000000            2.000000      595.000000
75%         4.000000    3395.000000            3.000000      949.000000
max       411.000000   27100.000000           20.000000    14700.000000
std         2.666546    1823.093688            1.630255      522.038648

       max_item_price  total_onshift_partners  total_busy_partners  \
count   197428.000000           197428.000000        197428.000000
mean      1159.588630               44.741531            41.678815
min          0.000000               -4.000000            -5.000000
25%        800.000000               19.000000            17.000000
50%       1095.000000               41.000000            39.000000
75%       1395.000000               62.000000            59.000000
max      14700.000000              171.000000           154.000000
std        558.411377               33.074996            30.794041

       total_outstanding_orders           hour            day
count             197428.000000  197428.000000  197428.000000
mean                  58.045941       8.467213       3.218966
min                   -6.000000       0.000000       0.000000
25%                   19.000000       2.000000       1.000000
50%                   47.000000       3.000000       3.000000
75%                   80.000000      19.000000       5.000000
max                  285.000000      23.000000       6.000000
std                   50.446361       8.658759       2.045789
```

[ ]:
```
categorical_columns = ['market_id', 'store_id', 'store_primary_category',␣
 ↪'order_protocol']
for i in categorical_columns:
  print(f"{i} : ",df[i].nunique())

# from this information we can drop store_id
# and keep market_id, store_primary_category and order_protocol for training␣
 ↪the neural network
# store_id might not add too much information for our neural network
# store_primary_category contains information about different types of␣
 ↪restaurant
# this might give us useful information and so we keep it
# order_protocol has less cardinality so we can use label encoding
# store_primary_category has vert high cardinality and so if we use label or␣
 ↪onehot encoding the number of column will increase drastically.so, we use
# frequency encoding
# for market_id we use label_encoding as it has low cardinality
```

```
market_id :  6
store_id :  6743
store_primary_category :  74
order_protocol :  7
```

[ ]:
```
df.rename(columns={
    'hour': 'created_hour',
    'day': 'created_day'
}, inplace=True)
# renaming the columns hour and day to created_hour and created_day respectively
```

[ ]:
```
df['time_for_delivery'] = df['actual_delivery_time'] - df['created_at']
# creating the column time_for_delivery by taking the difference of the columns␣
 ↪actual_delivery_time & created_at
# this gives us the time taken for the delivery
```

[ ]:
```
df['time_for_delivery']
```

[ ]:
```
0        0 days 01:02:59
1        0 days 01:07:04
2        0 days 00:29:41
3        0 days 00:51:15
4        0 days 00:39:50
                ...
197423   0 days 01:05:07
197424   0 days 00:56:23
197425   0 days 00:50:08
197426   0 days 01:05:07
197427   0 days 00:37:08
```

```
Name: time_for_delivery, Length: 197428, dtype: timedelta64[ns]
```

[ ]:
```
# dropping actual_delivery_time. why?
# the target variable here is time_for_delivery which is obtained by taking the␣
 ↪difference of actual_delivery_time & created_at
# If we do not drop the column time_for_delivery the target variable can be␣
 ↪easily obtained by subtracting the column created_at
# from actual_delivery_time making all other columns not so useful
# so we drop actual_delivery_time
df.drop(columns = ['actual_delivery_time','store_id'],inplace = True)
```

[ ]:
```
# dropping the create_at column as we've already extracted day and hour of the␣
 ↪week from this column
df.drop(columns = ['created_at'],inplace = True)
```

[ ]:
```
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()

# Label encode 'market_id'
df['market_id_encoded'] = label_encoder.fit_transform(df['market_id'])

# Label encode 'order_protocol'
df['order_protocol_encoded'] = label_encoder.fit_transform(df['order_protocol'])

# Frequency Encoding
frequency_encoding = df['store_primary_category'].value_counts().to_dict()
df['store_primary_category_encoded'] = df['store_primary_category'].
 ↪map(frequency_encoding)
```

[ ]:
```
df.head()
# we can now drop store_primary_category column as it has very high cardinality
```

[ ]:

	market_id	store_primary_category	order_protocol	total_items	subtotal	\
0	1.0	american	1.0	4	3441	
1	2.0	mexican	2.0	1	1900	
2	3.0	other	1.0	1	1900	
3	3.0	other	1.0	6	6900	
4	3.0	other	1.0	3	3900	

	num_distinct_items	min_item_price	max_item_price	total_onshift_partners	\
0	4	557.0	1239.0	33	
1	1	1400.0	1400.0	1	
2	1	1900.0	1900.0	1	
3	5	600.0	1800.0	1	
4	3	1100.0	1600.0	6	

```
   total_busy_partners  total_outstanding_orders  created_hour  created_day  \
0                   14                        21            22            4
1                    2                         2            21            1
2                    0                         0            20            3
3                    1                         2            21            1
4                    6                         9             2            6

  time_for_delivery  market_id_encoded  order_protocol_encoded  \
0   0 days 01:02:59                  0                       0
1   0 days 01:07:04                  1                       1
2   0 days 00:29:41                  2                       0
3   0 days 00:51:15                  2                       0
4   0 days 00:39:50                  2                       0

   store_primary_category_encoded
0                           19399
1                           17099
2                            8748
3                            8748
4                            8748
```

[ ]:
```
df.drop(columns=['store_primary_category'],inplace = True)
```

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

[ ]:
```
  market_id order_protocol  total_items  subtotal  num_distinct_items  \
0       1.0            1.0            4      3441                   4
1       2.0            2.0            1      1900                   1
2       3.0            1.0            1      1900                   1
3       3.0            1.0            6      6900                   5
4       3.0            1.0            3      3900                   3

   min_item_price  max_item_price  total_onshift_partners  \
0           557.0          1239.0                      33
1          1400.0          1400.0                       1
2          1900.0          1900.0                       1
3           600.0          1800.0                       1
4          1100.0          1600.0                       6

   total_busy_partners  total_outstanding_orders  created_hour  created_day  \
0                   14                        21            22            4
1                    2                         2            21            1
2                    0                         0            20            3
3                    1                         2            21            1
4                    6                         9             2            6

  time_for_delivery  market_id_encoded  order_protocol_encoded  \
```

```
0   0 days 01:02:59                 0                       0
1   0 days 01:07:04                 1                       1
2   0 days 00:29:41                 2                       0
3   0 days 00:51:15                 2                       0
4   0 days 00:39:50                 2                       0

   store_primary_category_encoded
0                           19399
1                           17099
2                            8748
3                            8748
4                            8748
```

[ ]:
```
# changing the target column to minutes
df['target'] = df['time_for_delivery'].dt.total_seconds() / 60
```

[ ]:
```
df.drop(columns=['market_id','order_protocol'],inplace = True)
```

[ ]:
```
df.rename(columns={'market_id_encoded':'market_id',
                   'order_protocol_encoded':'order_protocol',
                   'store_primary_category_encoded':
 ↪'store_primary_category'},inplace=True)
```

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

[ ]:
```
total_items                0
subtotal                   0
num_distinct_items         0
min_item_price             0
max_item_price             0
total_onshift_partners     0
total_busy_partners        0
total_outstanding_orders   0
created_hour               0
created_day                0
time_for_delivery          7
market_id                  0
order_protocol             0
store_primary_category     0
target                     7
dtype: int64
```

[ ]:
```
df.drop(columns=['time_for_delivery'],inplace=True)
```

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

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

[ ]:
```
total_items                 0
subtotal                    0
num_distinct_items          0
min_item_price              0
max_item_price              0
total_onshift_partners      0
total_busy_partners         0
total_outstanding_orders    0
created_hour                0
created_day                 0
market_id                   0
order_protocol              0
store_primary_category      0
target                      0
dtype: int64
```

[ ]:
```
df.duplicated().sum()
```

[ ]:
```
0
```

[ ]:
```
df.drop(columns=['store_primary_category'],inplace=True)
# dropping store_primary_category since it has very high cardinality
```

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

```
<class 'pandas.core.frame.DataFrame'>
Index: 197421 entries, 0 to 197427
Data columns (total 13 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   total_items               197421 non-null  int64
 1   subtotal                  197421 non-null  int64
 2   num_distinct_items        197421 non-null  int64
 3   min_item_price            197421 non-null  float64
 4   max_item_price            197421 non-null  float64
 5   total_onshift_partners    197421 non-null  int64
 6   total_busy_partners       197421 non-null  int64
 7   total_outstanding_orders  197421 non-null  int64
 8   created_hour              197421 non-null  int32
 9   created_day               197421 non-null  int32
 10  market_id                 197421 non-null  int64
 11  order_protocol            197421 non-null  int64
 12  target                    197421 non-null  float64
dtypes: float64(3), int32(2), int64(8)
memory usage: 19.6 MB
```

[ ]:
```
# Data preprocessing
y=df['target']
x=df.drop(['target'],axis=1)
```

[ ]:
```
from sklearn.model_selection import train_test_split

X_train,X_test,y_train,y_test=train_test_split(x,y,test_size=0.
 ↪2,random_state=42)
```

[ ]:
```
x.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 197421 entries, 0 to 197427
Data columns (total 12 columns):
 #   Column                   Non-Null Count   Dtype
---  ------                   --------------   -----
 0   total_items              197421 non-null  int64
 1   subtotal                 197421 non-null  int64
 2   num_distinct_items       197421 non-null  int64
 3   min_item_price           197421 non-null  float64
 4   max_item_price           197421 non-null  float64
 5   total_onshift_partners   197421 non-null  int64
 6   total_busy_partners      197421 non-null  int64
 7   total_outstanding_orders 197421 non-null  int64
 8   created_hour             197421 non-null  int32
 9   created_day              197421 non-null  int32
 10  market_id                197421 non-null  int64
 11  order_protocol           197421 non-null  int64
dtypes: float64(2), int32(2), int64(8)
memory usage: 18.1 MB
```

[ ]:
```
x.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 197421 entries, 0 to 197427
Data columns (total 12 columns):
 #   Column                   Non-Null Count   Dtype
---  ------                   --------------   -----
 0   total_items              197421 non-null  int64
 1   subtotal                 197421 non-null  int64
 2   num_distinct_items       197421 non-null  int64
 3   min_item_price           197421 non-null  float64
 4   max_item_price           197421 non-null  float64
 5   total_onshift_partners   197421 non-null  int64
 6   total_busy_partners      197421 non-null  int64
 7   total_outstanding_orders 197421 non-null  int64
 8   created_hour             197421 non-null  int32
 9   created_day              197421 non-null  int32
```

```
 10  market_id                 197421 non-null  int64
 11  order_protocol            197421 non-null  int64
dtypes: float64(2), int32(2), int64(8)
memory usage: 18.1 MB
```

[ ]:
```
from sklearn import preprocessing
import pandas as pd

# scaling the data

scaler=preprocessing.MinMaxScaler()

x_scaled=scaler.fit_transform(x)

X_train,X_test,y_train,y_test=train_test_split(x_scaled,y,test_size=0.
 ↪2,random_state=42)
```

[ ]:
```
# We will build a simple neural network to train our regression model it is a␣
 ↪sequential model with two layers,
# we have kept the number of nodes in the first layers equal to the number of␣
 ↪input columns, and for the subsequent layers 32, 32, which can we changed or␣
 ↪experimented with
# the activation for the layers is kept as relu because it is a great non␣
 ↪linear activation function that works for most cases, we could have used␣
 ↪leaky relu if we see gradient vanishing.
# the last layer has one node because it will give the single result that is␣
 ↪our delivery time and the activation function for that should be linear
```

[ ]:
```
X_train.shape
```

[ ]:
```
(157936, 12)
```

[ ]:
```
x.columns
```

[ ]:
```
Index(['total_items', 'subtotal', 'num_distinct_items', 'min_item_price',
       'max_item_price', 'total_onshift_partners', 'total_busy_partners',
       'total_outstanding_orders', 'created_hour', 'created_day', 'market_id',
       'order_protocol'],
      dtype='object')
```

[ ]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

[ ]:
```
from sklearn.metrics import mean_squared_error,mean_absolute_error,r2_score
```

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

```
# Check for NaN values
nan_count = np.isnan(X_train).sum()
print(f"Number of NaN values in X_train: {nan_count}")

# Check for infinite values
inf_count = np.isinf(X_train).sum()
print(f"Number of infinite values in X_train: {inf_count}")

# Similarly, we can check for y_train
nan_count_y = np.isnan(y_train).sum()
print(f"Number of NaN values in y_train: {nan_count_y}")

inf_count_y = np.isinf(y_train).sum()
print(f"Number of infinite values in y_train: {inf_count_y}")
```

```
Number of NaN values in X_train: 0
Number of infinite values in X_train: 0
Number of NaN values in y_train: 0
Number of infinite values in y_train: 0
```

[ ]:
```
'''
# dropping the NaN values in y_train
mask = ~np.isnan(y_train)

# Filter out NaN values
X_train = X_train[mask]
y_train = y_train[mask]

# Checking if NaNs are removed
print(f"Number of NaN values in y_train_clean: {np.isnan(y_train_clean).sum()}")
'''
```

[ ]:
```
'\n# dropping the NaN values in y_train\nmask = ~np.isnan(y_train)\n\n# Filter
out NaN values\nX_train = X_train[mask]\ny_train = y_train[mask]\n\n# Checking
if NaNs are removed\nprint(f"Number of NaN values in y_train_clean:
{np.isnan(y_train_clean).sum()}")\n'
```

[ ]:
```
# We will build a simple neural network to train our regression model it is a␣
 ↪sequential model with two layers,
# we have kept the number of nodes in the first layers equal to the number of␣
 ↪input columns, and for the subsequent layers 32, 32, which can we changed or␣
 ↪experimented with
# the activation for the layers is kept as relu because it is a great non␣
 ↪linear activation function that works for most cases, we could have used␣
 ↪leaky relu if we see gradient vanishing.
# the last layer has one node because it will give the single result that is␣
 ↪our delivery time and the activation function for that should be linear
```

[ ]:
```
model=Sequential()
model.add(Dense(12,kernel_initializer='normal',activation='relu'))
model.add(Dense(32,activation='relu'))
model.add(Dense(32,activation='relu'))
model.add(Dense(1,activation='linear'))
model.compile(loss='mse',optimizer='Adam',metrics=['mse','mae'])
history=model.
  ↪fit(X_train,y_train,epochs=20,batch_size=512,verbose=1,validation_split=0.2)
```

```
Epoch 1/20
247/247             3s 3ms/step -
loss: 23321.8105 - mae: 40.3260 - mse: 23321.8105 - val_loss: 453.9595 -
val_mae: 15.7035 - val_mse: 453.9595
Epoch 2/20
247/247             1s 3ms/step -
loss: 292204.5938 - mae: 17.1139 - mse: 292204.5938 - val_loss: 416.0326 -
val_mae: 14.2983 - val_mse: 416.0326
Epoch 3/20
247/247             1s 3ms/step -
loss: 32966.0508 - mae: 14.4582 - mse: 32966.0508 - val_loss: 399.6105 -
val_mae: 14.0052 - val_mse: 399.6105
Epoch 4/20
247/247             1s 3ms/step -
loss: 83243.9062 - mae: 14.5458 - mse: 83243.9062 - val_loss: 390.2655 -
val_mae: 13.8043 - val_mse: 390.2655
Epoch 5/20
247/247             1s 3ms/step -
loss: 323720.0625 - mae: 16.6165 - mse: 323720.0625 - val_loss: 382.6169 -
val_mae: 13.6709 - val_mse: 382.6169
Epoch 6/20
247/247             1s 3ms/step -
loss: 262877.1562 - mae: 15.7771 - mse: 262877.1562 - val_loss: 376.4873 -
val_mae: 13.5950 - val_mse: 376.4873
Epoch 7/20
247/247             2s 4ms/step -
loss: 15172.6416 - mae: 13.6098 - mse: 15172.6416 - val_loss: 403.0890 -
val_mae: 15.1243 - val_mse: 403.0890
Epoch 8/20
247/247             1s 4ms/step -
loss: 604468.8125 - mae: 18.8247 - mse: 604468.8125 - val_loss: 368.2538 -
val_mae: 13.4595 - val_mse: 368.2538
Epoch 9/20
247/247             1s 5ms/step -
loss: 24642.6465 - mae: 13.5975 - mse: 24642.6465 - val_loss: 372.9906 -
val_mae: 13.9618 - val_mse: 372.9906
Epoch 10/20
247/247             1s 4ms/step -
```

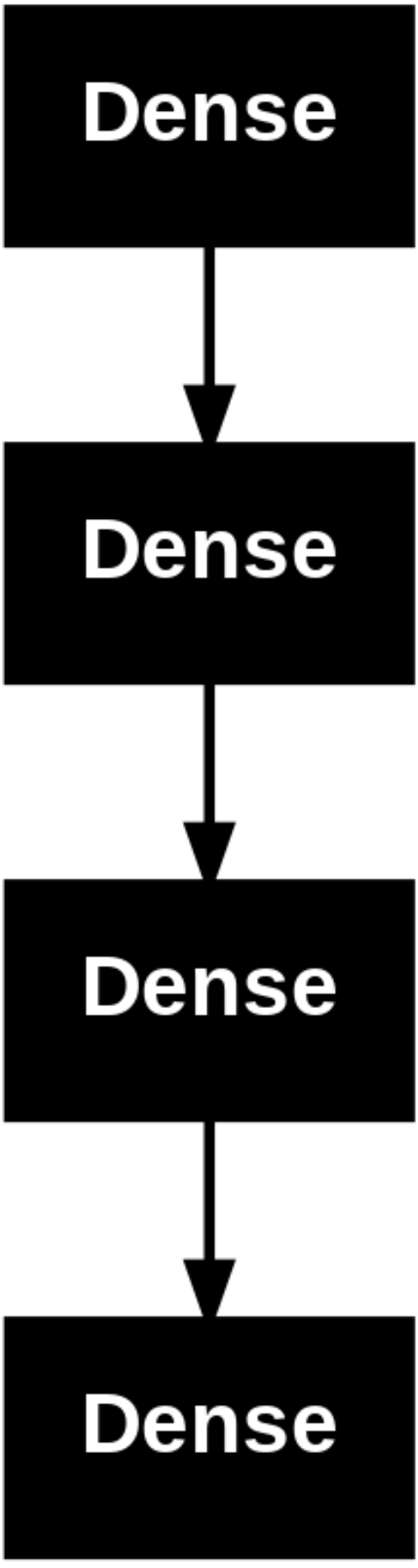
```
loss: 212253.7812 - mae: 15.2923 - mse: 212253.7812 - val_loss: 363.3054 -
val_mae: 13.3337 - val_mse: 363.3054
Epoch 11/20
247/247             1s 2ms/step -
loss: 42523.0312 - mae: 13.6236 - mse: 42523.0312 - val_loss: 364.7053 -
val_mae: 13.3922 - val_mse: 364.7053
Epoch 12/20
247/247             1s 3ms/step -
loss: 149347.5000 - mae: 14.6400 - mse: 149347.5000 - val_loss: 359.9142 -
val_mae: 13.3389 - val_mse: 359.9142
Epoch 13/20
247/247             1s 3ms/step -
loss: 46731.1016 - mae: 13.5380 - mse: 46731.1016 - val_loss: 361.3734 -
val_mae: 13.3929 - val_mse: 361.3734
Epoch 14/20
247/247             1s 3ms/step -
loss: 142485.2656 - mae: 14.4994 - mse: 142485.2656 - val_loss: 357.4255 -
val_mae: 13.3103 - val_mse: 357.4255
Epoch 15/20
247/247             1s 3ms/step -
loss: 104481.1562 - mae: 14.2247 - mse: 104481.1562 - val_loss: 356.5236 -
val_mae: 13.3096 - val_mse: 356.5236
Epoch 16/20
247/247             1s 3ms/step -
loss: 241224.8750 - mae: 15.3501 - mse: 241224.8750 - val_loss: 353.9882 -
val_mae: 13.1757 - val_mse: 353.9882
Epoch 17/20
247/247             1s 3ms/step -
loss: 200180.1250 - mae: 14.7885 - mse: 200180.1250 - val_loss: 352.7170 -
val_mae: 13.2105 - val_mse: 352.7170
Epoch 18/20
247/247             1s 3ms/step -
loss: 403935.8438 - mae: 16.4911 - mse: 403935.8438 - val_loss: 351.1131 -
val_mae: 13.1447 - val_mse: 351.1131
Epoch 19/20
247/247             1s 3ms/step -
loss: 14423.3066 - mae: 13.1948 - mse: 14423.3066 - val_loss: 381.5750 -
val_mae: 14.7792 - val_mse: 381.5750
Epoch 20/20
247/247             2s 4ms/step -
loss: 140783.4688 - mae: 14.5393 - mse: 140783.4688 - val_loss: 352.5788 -
val_mae: 13.2003 - val_mse: 352.5788
```

[ ]:
```
model.summary()
from tensorflow.keras.utils import plot_model
plot_model(model)
```

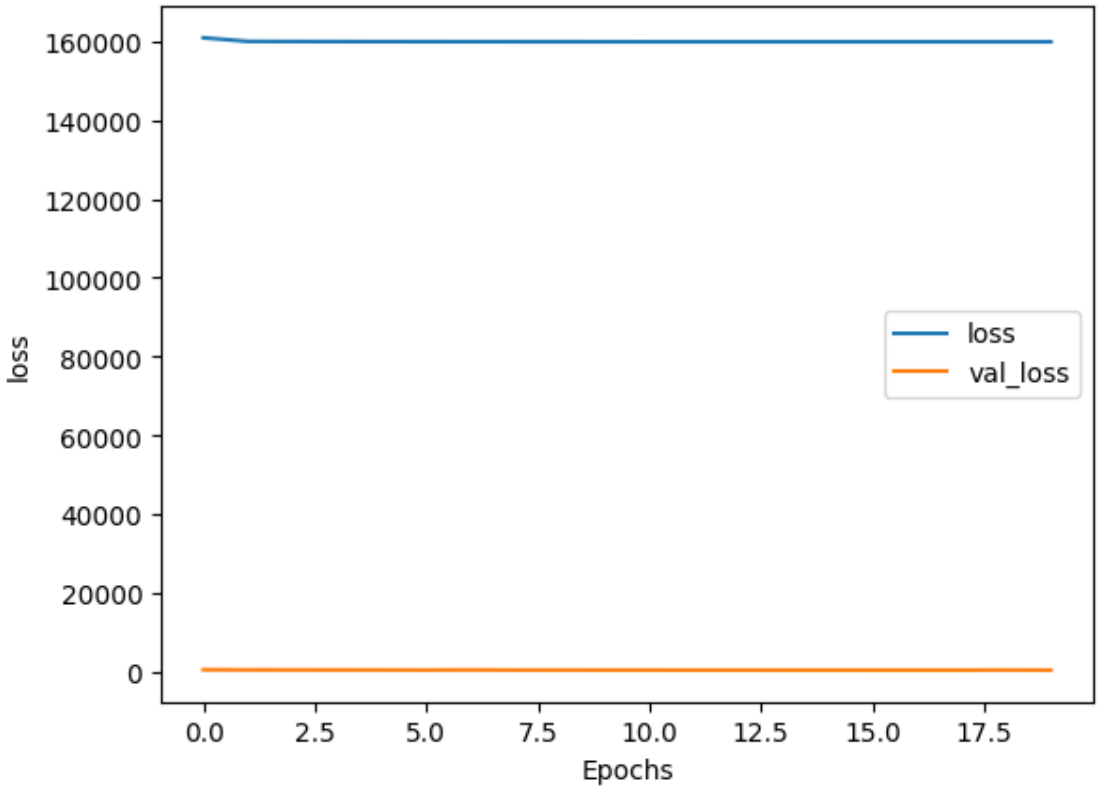
```
Model: "sequential_12"
```

```
Layer (type)                                Output Shape                              ␣
↪Param #

dense_48 (Dense)                            (None, 12)                                ␣
↪156

dense_49 (Dense)                            (None, 32)                                ␣
↪416

dense_50 (Dense)                            (None, 32)                                ␣
↪1,056

dense_51 (Dense)                            (None, 1)                                 ␣
↪ 33


Total params: 4,985 (19.48 KB)

Trainable params: 1,661 (6.49 KB)

Non-trainable params: 0 (0.00 B)

Optimizer params: 3,324 (12.99 KB)
```

[ ]:


Dense
Dense
Dense
Dense

[ ]:
```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.xlabel("Epochs")
plt.ylabel('loss')
plt.legend(['loss','val_loss'])
plt.show()
```



[ ]:
```
print('r2_score:',r2_score(y_test, model.predict(X_test)))
mse = mean_squared_error(y_test, model.predict(X_test))
rmse = mse**.5
print("mse : ",mse)
print("rmse : ",rmse)
print("errors for neural net")
mae = mean_absolute_error(y_test, model.predict(X_test))
print("mae : ",mae)
```

```
1234/1234              2s 1ms/step
r2_score: 0.015107730969296518
1234/1234              2s 1ms/step
```

```
mse :  1330.890268407533
rmse :  36.48136878473083
errors for neural net
1234/1234             2s 1ms/step
mae :  13.219205568283085
```

[ ]:
```
from sklearn.metrics import mean_absolute_percentage_error
print("Mean Absolute Percentage Error :␣
  ↪",mean_absolute_percentage_error(y_test, model.predict(X_test)))
# we can see the Mean Absolute Error
```

```
1234/1234             3s 2ms/step
Mean Absolute Percentage Error :  0.3045003462420059
```