Problem Statement

Porter is India's Largest Marketplace for Intra-City Logistics. Leader in the country's $40 billion intra-city logistics market, Porter strives to improve the lives of 1,50,000+ driver-partners by providing them with consistent earning & independence. Currently, the company has serviced 5+ million customers

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

Loading Library

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.impute import KNNImputer
```

Loading file from link

```
import requests

# URL of the ZIP file
url = "https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/015/039/original/dataset.csv.zip?1663710760"

# Local file path to save the downloaded ZIP
local_filename = "portor.zip"

# Send a GET request to the URL
response = requests.get(url, stream=True)

# Write the content to a local file
with open(local_filename, "wb") as file:
    for chunk in response.iter_content(chunk_size=8192):
        file.write(chunk)

print(f"File downloaded successfully as {local_filename}")
```

```
File downloaded successfully as portor.zip
```

```
import zipfile

# Path to the ZIP file
zip_file_path = "portor.zip"

# Destination folder to extract contents
extract_to_folder = "extracted_files"

# Open the ZIP file and extract all contents
with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
    zip_ref.extractall(extract_to_folder)

print(f"Files extracted to {extract_to_folder}")
```

```
Files extracted to extracted_files
```

```
df = pd.read_csv("extracted_files/dataset.csv")
```

```
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
```

Converting columns to required data type for below columns

```
df['created_at'] = pd.to_datetime(df['created_at'])
df['actual_delivery_time'] = pd.to_datetime(df['actual_delivery_time'])
df['market_id'] = df['market_id'].astype('category')
df['order_protocol'] = df['order_protocol'].astype('category')
df['store_primary_category'] = df['store_primary_category'].astype('category')
df['day_create'] = df['created_at'].dt.day
df['month_create'] = df['created_at'].dt.month
df['year_create'] = df['created_at'].dt.year
df['day_of_wk_create'] = df['created_at'].dt.dayofweek
df['hr_create'] = df['created_at'].dt.hour
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 197428 entries, 0 to 197427
Data columns (total 19 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   market_id                 196441 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  object
 4   store_primary_category    192668 non-null  category
 5   order_protocol            196433 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  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
 14  day_create                197428 non-null  int32
 15  month_create              197428 non-null  int32
 16  year_create               197428 non-null  int32
 17  day_of_wk_create          197428 non-null  int32
 18  hr_create                 197428 non-null  int32
dtypes: category(3), datetime64[ns](2), float64(3), int32(5), int64(5), object(1)
memory usage: 20.9+ MB
```

```
# Creating target columns in hours (difference in hours)
df['tar_est_time'] = (df['actual_delivery_time'] - df['created_at']).dt.total_seconds() / 3600
df['tar_est_time'].head()
```

	tar_est_time
0	1.049722
1	1.117778
2	0.494722
3	0.854167
4	0.663889

dtype: float64

EDA

- 6 columns having null values--> total_onshift_partners, total_busy_partners, total_outstanding_orders, store_primary_category, market_id
- total of 197428 records are available
- 14 columns are available
- 4 columns have data type as 'object' and columns are menetioned below created_at, actual_delivery_time, store_id, store_primary_category
- day, month, hours, day and week of day columns are created.
- 2014-10-19 to 2015-02-18 are range of dates from created_id columns

```
df.head()
```

```
df.describe()
```

	created_at	actual_delivery_time	total_items	subtotal	num_distinct_items	min_item_price	max_item_price
count	197428	197421	197428.000000	197428.000000	197428.000000	197428.000000	197428.000000
mean	2015-02-04 22:00:09.537962752	2015-02-04 22:48:23.348914432	3.196391	2682.331402	2.670791	686.218470	1159.588630
min	2014-10-19 05:24:15	2015-01-21 15:58:11	1.000000	0.000000	1.000000	-86.000000	0.000000
25%	2015-01-29 02:32:42	2015-01-29 03:22:29	2.000000	1400.000000	1.000000	299.000000	800.000000
50%	2015-02-05 03:29:09.500000	2015-02-05 04:40:41	3.000000	2200.000000	2.000000	595.000000	1095.000000
75%	2015-02-12 01:39:18.500000	2015-02-12 02:25:26	4.000000	3395.000000	3.000000	949.000000	1395.000000
max	2015-02-18 06:00:44	2015-02-19 22:45:31	411.000000	27100.000000	20.000000	14700.000000	14700.000000
std	NaN	NaN	2.666546	1823.093688	1.630255	522.038648	558.411377

```
round(df.isnull().sum()/df.shape[0]*100,2).sort_values(ascending=False)
```

	0
total_busy_partners	8.24
total_outstanding_orders	8.24
total_onshift_partners	8.24
store_primary_category	2.41
order_protocol	0.50
market_id	0.50
created_at	0.00
actual_delivery_time	0.00
store_id	0.00
num_distinct_items	0.00
subtotal	0.00
total_items	0.00
max_item_price	0.00
min_item_price	0.00
day_create	0.00
month_create	0.00
year_create	0.00
day_of_wk_create	0.00
hr_create	0.00
tar_est_time	0.00

dtype: float64

```
df.shape
```

```
(197428, 20)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 197428 entries, 0 to 197427
Data columns (total 20 columns):
 #   Column                   Non-Null Count   Dtype
---  ------                   --------------   -----
 0   market_id                196441 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  object
 4   store_primary_category   192668 non-null  category
 5   order_protocol           196433 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  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
 14  day_create               197428 non-null  int32
 15  month_create             197428 non-null  int32
 16  year_create              197428 non-null  int32
 17  day_of_wk_create         197428 non-null  int32
 18  hr_create                197428 non-null  int32
 19  tar_est_time             197421 non-null  float64
dtypes: category(3), datetime64[ns](2), float64(4), int32(5), int64(5), object(1)
memory usage: 22.4+ MB
```

Univariate Analysis

Observation found -

- market_id 2 is count is more
- Order protorcal 1 and 3, count is high

- Create_id column, day 18, 19 and 20th are very minimal or data is missing on those days.
- only Jan, feb and Oct data are available
- Majority data are from 2015.
- Sat and sunday have majority of orders
- 2 hr and 20hr are peak of orders
- orders are more focued on early in moring and late in evening. Afternoon orders are minimum.
- All numerical columns are right skewed.
- All numerical columns have outliers and outlier are removed later in code

```
col = df.columns
for i in col:
    print('Unique values of',i, 'column =', df[i].nunique())

    print('-----------------------------------------------')
```

```
Unique values of market_id column = 6
-----------------------------------------------
Unique values of created_at column = 180985
-----------------------------------------------
Unique values of actual_delivery_time column = 178110
-----------------------------------------------
Unique values of store_id column = 6743
-----------------------------------------------
Unique values of store_primary_category column = 74
-----------------------------------------------
Unique values of order_protocol column = 7
-----------------------------------------------
Unique values of total_items column = 57
-----------------------------------------------
Unique values of subtotal column = 8368
-----------------------------------------------
Unique values of num_distinct_items column = 20
-----------------------------------------------
Unique values of min_item_price column = 2312
-----------------------------------------------
Unique values of max_item_price column = 2652
-----------------------------------------------
Unique values of total_onshift_partners column = 172
-----------------------------------------------
Unique values of total_busy_partners column = 159
-----------------------------------------------
Unique values of total_outstanding_orders column = 281
-----------------------------------------------
Unique values of day_create column = 30
-----------------------------------------------
Unique values of month_create column = 3
-----------------------------------------------
Unique values of year_create column = 2
-----------------------------------------------
Unique values of day_of_wk_create column = 7
-----------------------------------------------
Unique values of hr_create column = 19
-----------------------------------------------
Unique values of tar_est_time column = 7134
-----------------------------------------------
```

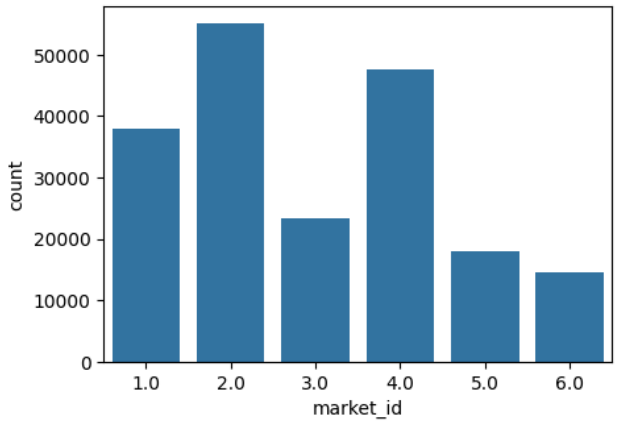
```
col = df.columns
for i in col:
    print('Unique values of',i, 'column')
    print((df[i].value_counts().head(3)/df.shape[0]) * 100)
    print('-----------------------------------------------')
```

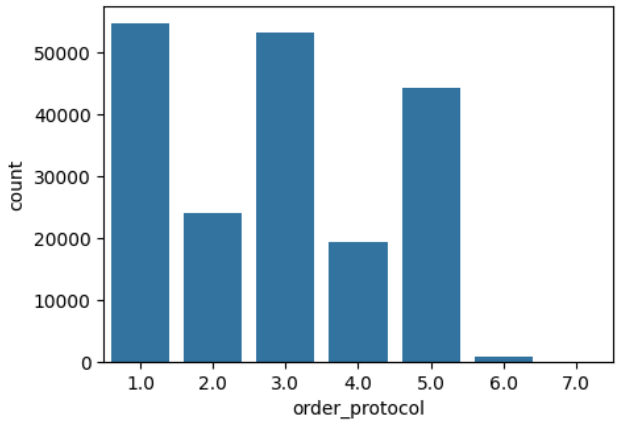
```
day_create
7     4.634094
15    4.602691
14    4.566728
Name: count, dtype: float64
------------------------------------------------
Unique values of month_create column
month_create
2     65.312418
1     34.687076
10     0.000507
Name: count, dtype: float64
------------------------------------------------
Unique values of year_create column
year_create
2015    99.999493
2014     0.000507
Name: count, dtype: float64
------------------------------------------------
Unique values of day_of_wk_create column
day_of_wk_create
5    17.495492
6    17.028993
4    14.119071
Name: count, dtype: float64
------------------------------------------------
Unique values of hr_create column
hr_create
2    18.728853
1    14.278623
3    13.710315
Name: count, dtype: float64
------------------------------------------------
Unique values of tar_est_time column
tar_est_time
0.689722    0.064834
0.633333    0.060275
0.633611    0.059769
Name: count, dtype: float64
------------------------------------------------
```

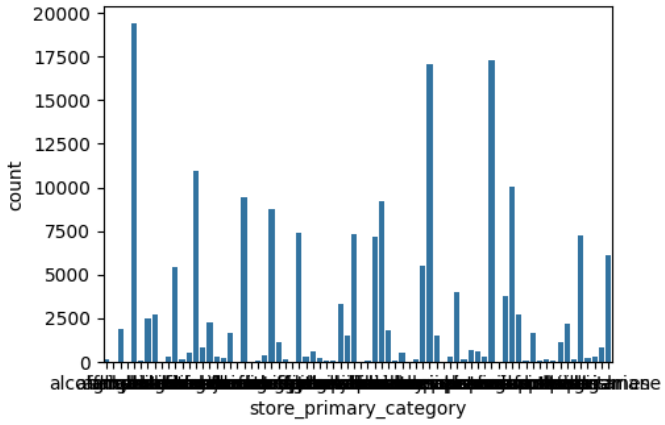
```
df.columns
```

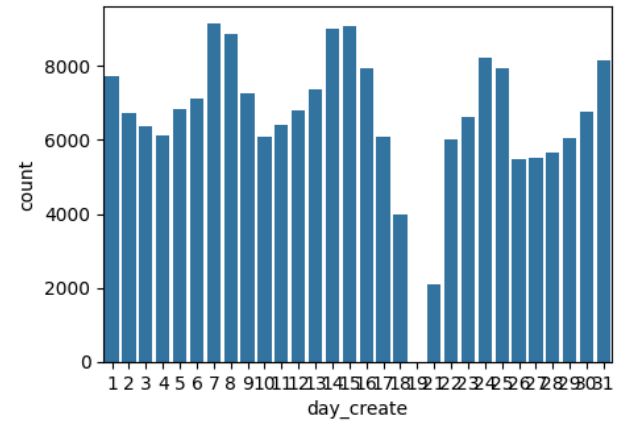
```
Index(['market_id', 'created_at', 'actual_delivery_time', 'store_id',
       'store_primary_category', 'order_protocol', 'total_items', 'subtotal',
       'num_distinct_items', 'min_item_price', 'max_item_price',
       'total_onshift_partners', 'total_busy_partners',
       'total_outstanding_orders', 'day_create', 'month_create', 'year_create',
       'day_of_wk_create', 'hr_create', 'tar_est_time'],
      dtype='object')
```

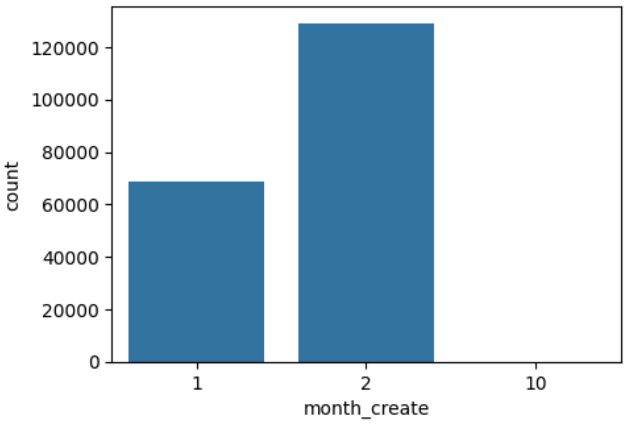
```
cat_cols = ['market_id', 'order_protocol', 'store_primary_category','day_create', 'month_create', 'year_create', 'day_of_wk_create',

fig, ax = plt.subplots(3,3, figsize = (15,10))

sns.countplot(ax = ax[0,0], x = df[cat_cols[0]])
sns.countplot(ax = ax[0,1], x = df[cat_cols[1]])
sns.countplot(ax = ax[0,2], x = df[cat_cols[2]])
sns.countplot(ax = ax[1,0], x = df[cat_cols[3]])
sns.countplot(ax = ax[1,1], x = df[cat_cols[4]])
sns.countplot(ax = ax[1,2], x = df[cat_cols[5]])
sns.countplot(ax = ax[2,0], x = df[cat_cols[6]])
sns.countplot(ax = ax[2,1], x = df[cat_cols[7]])
fig.delaxes(ax[2, 2])

plt.tight_layout()
plt.show()
```

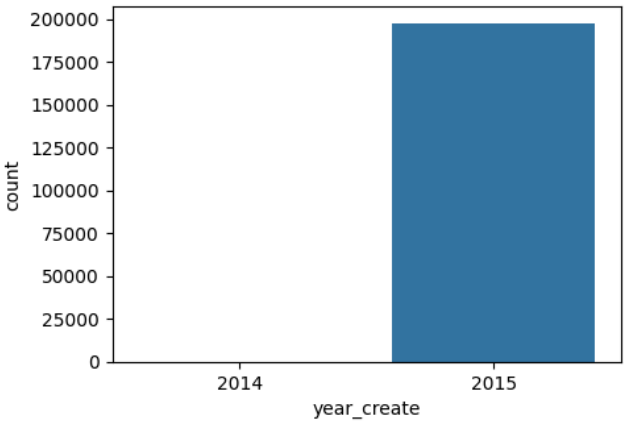


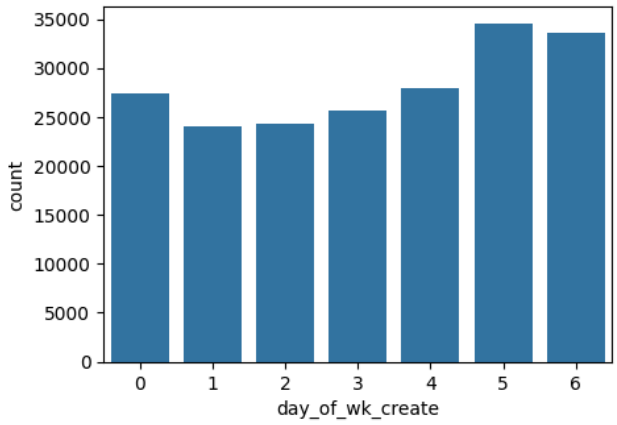


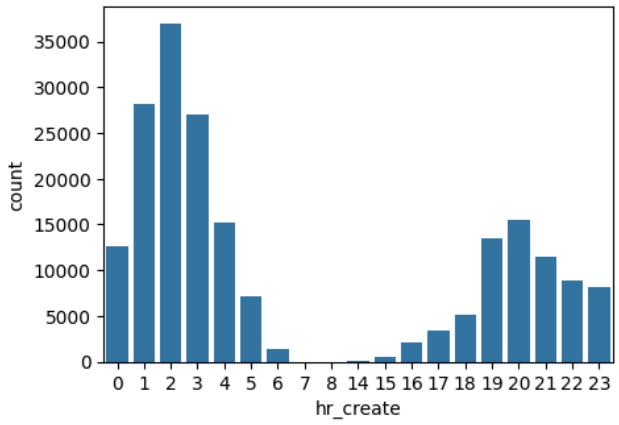




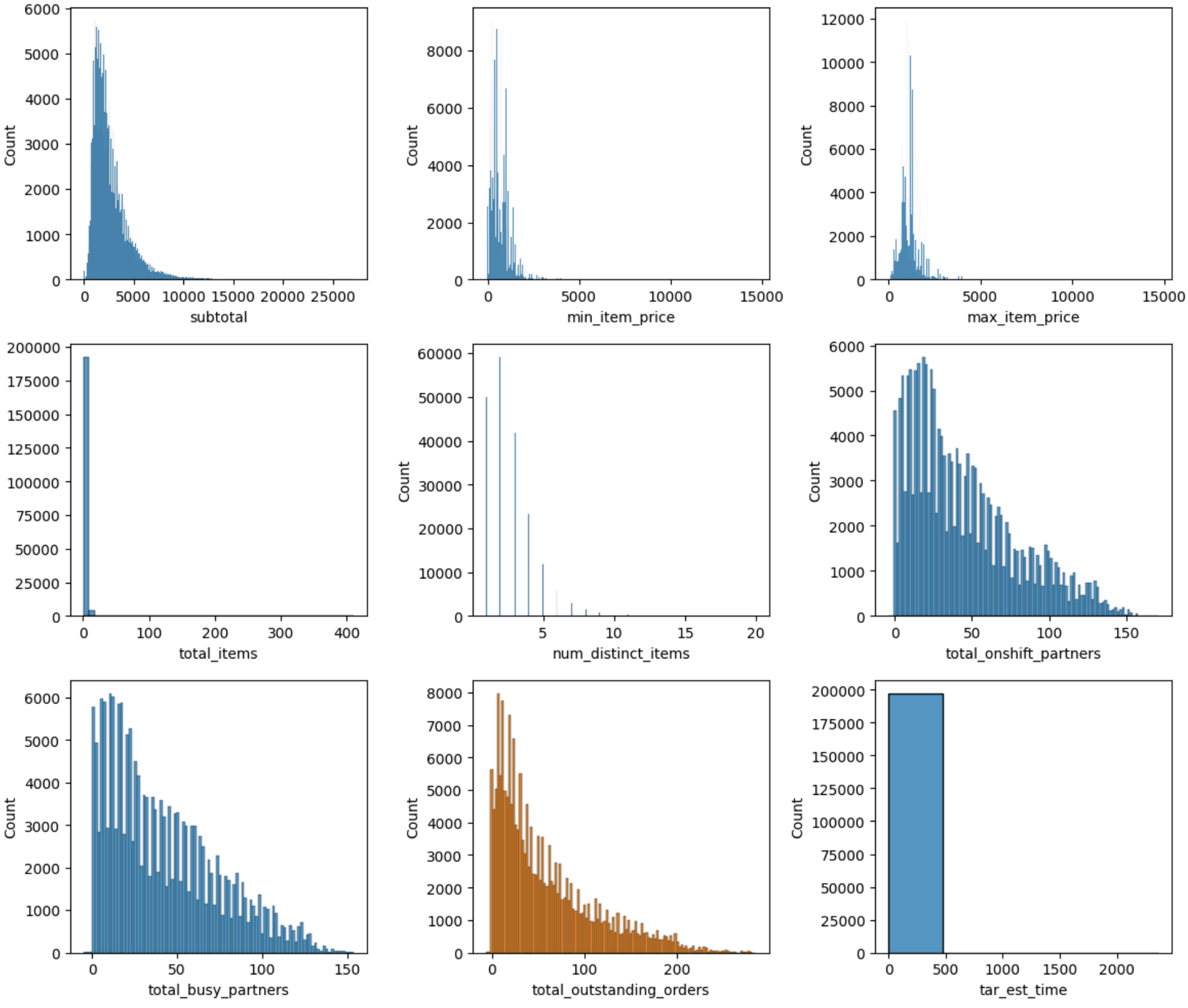








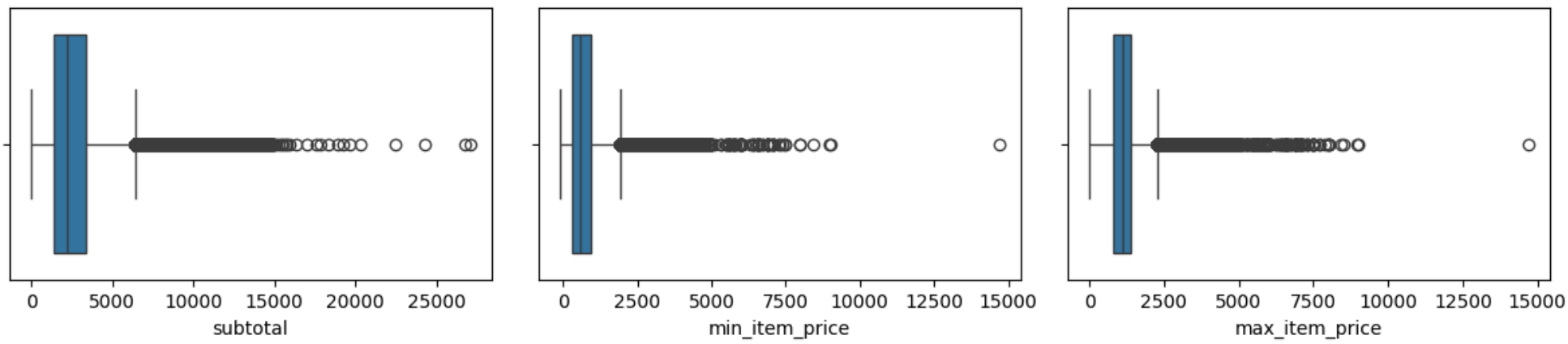

```
num_cols = ['subtotal', 'min_item_price', 'max_item_price', 'total_items', 'num_distinct_items', 'total_onshift_partners', 'total_bu
       'total_outstanding_orders', "tar_est_time"]

fig, ax = plt.subplots(3,3, figsize = (12,10))

sns.histplot(ax = ax[0,0], x = df[num_cols[0]])
sns.histplot(ax = ax[0,1], x = df[num_cols[1]])
sns.histplot(ax = ax[0,2], x = df[num_cols[2]])
sns.histplot(ax = ax[1,0], x = df[num_cols[3]], bins=50)
sns.histplot(ax = ax[1,1], x = df[num_cols[4]])
sns.histplot(ax = ax[1,2], x = df[num_cols[5]])
sns.histplot(ax = ax[2,0], x = df[num_cols[6]])
sns.histplot(ax = ax[2,1], x = df[num_cols[7]])
sns.histplot(ax = ax[2,1], x = df[num_cols[7]])
sns.histplot(ax = ax[2,2], x = df[num_cols[8]], bins=5)
#fig.delaxes(ax[2, 2])

plt.tight_layout()
plt.show()
```



Outlier detection:

- below mentioned columns have outliers

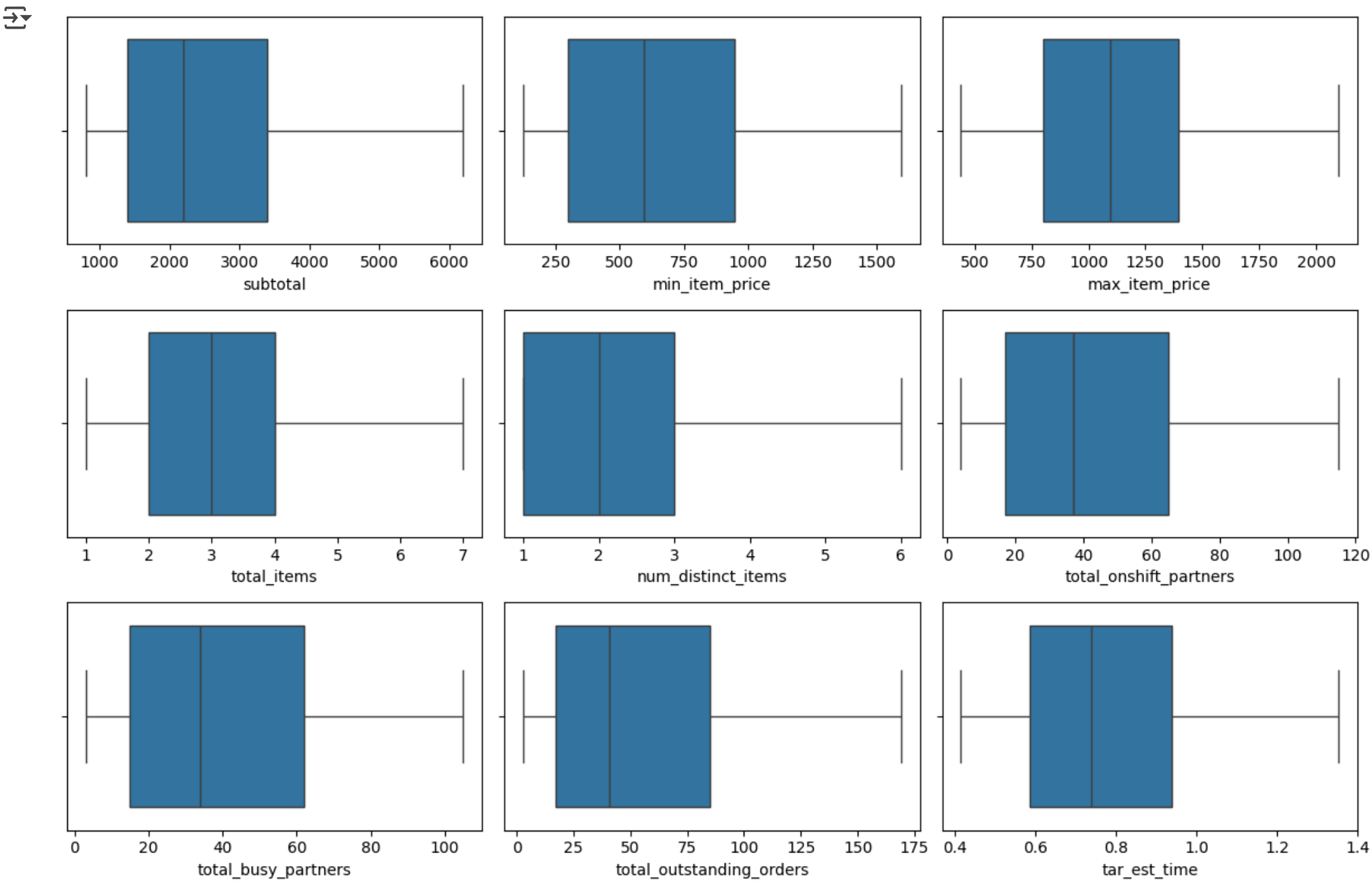
```
num_cols = ['subtotal', 'min_item_price', 'max_item_price', 'total_items', 'num_distinct_items', 'total_onshift_partners', 'total_bu
       'total_outstanding_orders', 'tar_est_time']

fig, ax = plt.subplots(3,3, figsize = (12,8))

sns.boxplot(ax = ax[0,0], x = df[num_cols[0]])
sns.boxplot(ax = ax[0,1], x = df[num_cols[1]])
sns.boxplot(ax = ax[0,2], x = df[num_cols[2]])
sns.boxplot(ax = ax[1,0], x = df[num_cols[3]])
sns.boxplot(ax = ax[1,1], x = df[num_cols[4]])
sns.boxplot(ax = ax[1,2], x = df[num_cols[5]])
sns.boxplot(ax = ax[2,0], x = df[num_cols[6]])
sns.boxplot(ax = ax[2,1], x = df[num_cols[7]])
sns.boxplot(ax = ax[2,2], x = df[num_cols[8]])
#fig.delaxes(ax[2, 2])

plt.tight_layout()
plt.show()
```



Removing outlier in each column

```
num_cols = ['subtotal', 'min_item_price', 'max_item_price', 'total_items', 'num_distinct_items', 'total_onshift_partners', 'total_bu
       'total_outstanding_orders','tar_est_time']

for i in num_cols:
  df[i] = df[i].clip(lower=df[i].quantile(0.05), upper=df[i].quantile(0.95))
```

```
num_cols = ['subtotal', 'min_item_price', 'max_item_price', 'total_items', 'num_distinct_items', 'total_onshift_partners', 'total_bu
       'total_outstanding_orders','tar_est_time']

fig, ax = plt.subplots(3,3, figsize = (12,8))

sns.boxplot(ax = ax[0,0], x = df[num_cols[0]])
sns.boxplot(ax = ax[0,1], x = df[num_cols[1]])
sns.boxplot(ax = ax[0,2], x = df[num_cols[2]])
sns.boxplot(ax = ax[1,0], x = df[num_cols[3]])
sns.boxplot(ax = ax[1,1], x = df[num_cols[4]])
sns.boxplot(ax = ax[1,2], x = df[num_cols[5]])
sns.boxplot(ax = ax[2,0], x = df[num_cols[6]])
sns.boxplot(ax = ax[2,1], x = df[num_cols[7]])
sns.boxplot(ax = ax[2,2], x = df[num_cols[8]])
#fig.delaxes(ax[2, 2])

plt.tight_layout()
plt.show()
```




```
df['tar_est_time'].describe()
```

	tar_est_time
count	197421.000000
mean	0.782268
std	0.256451
min	0.414444
25%	0.584444
50%	0.738889
75%	0.939167
max	1.353333

dtype: float64

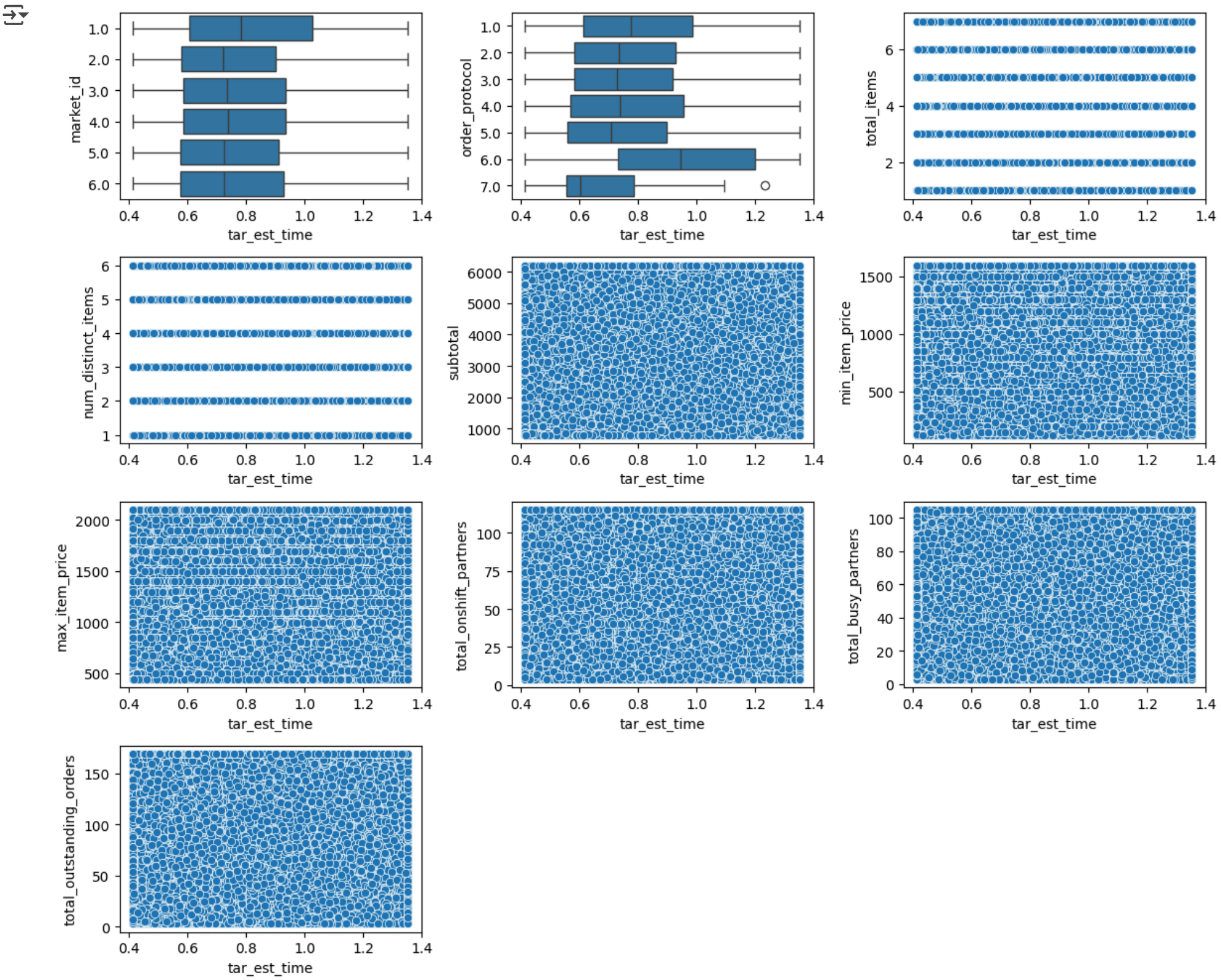
Binary Variate Analysis

Observation -

- Order protocal columns 6id took more delivery time when compared to others
- New distinct items have high correlation with total itmss and subtotal, vice versa
- Total onsite partners, total busy partners and total outstanding orders have high correlation with each other

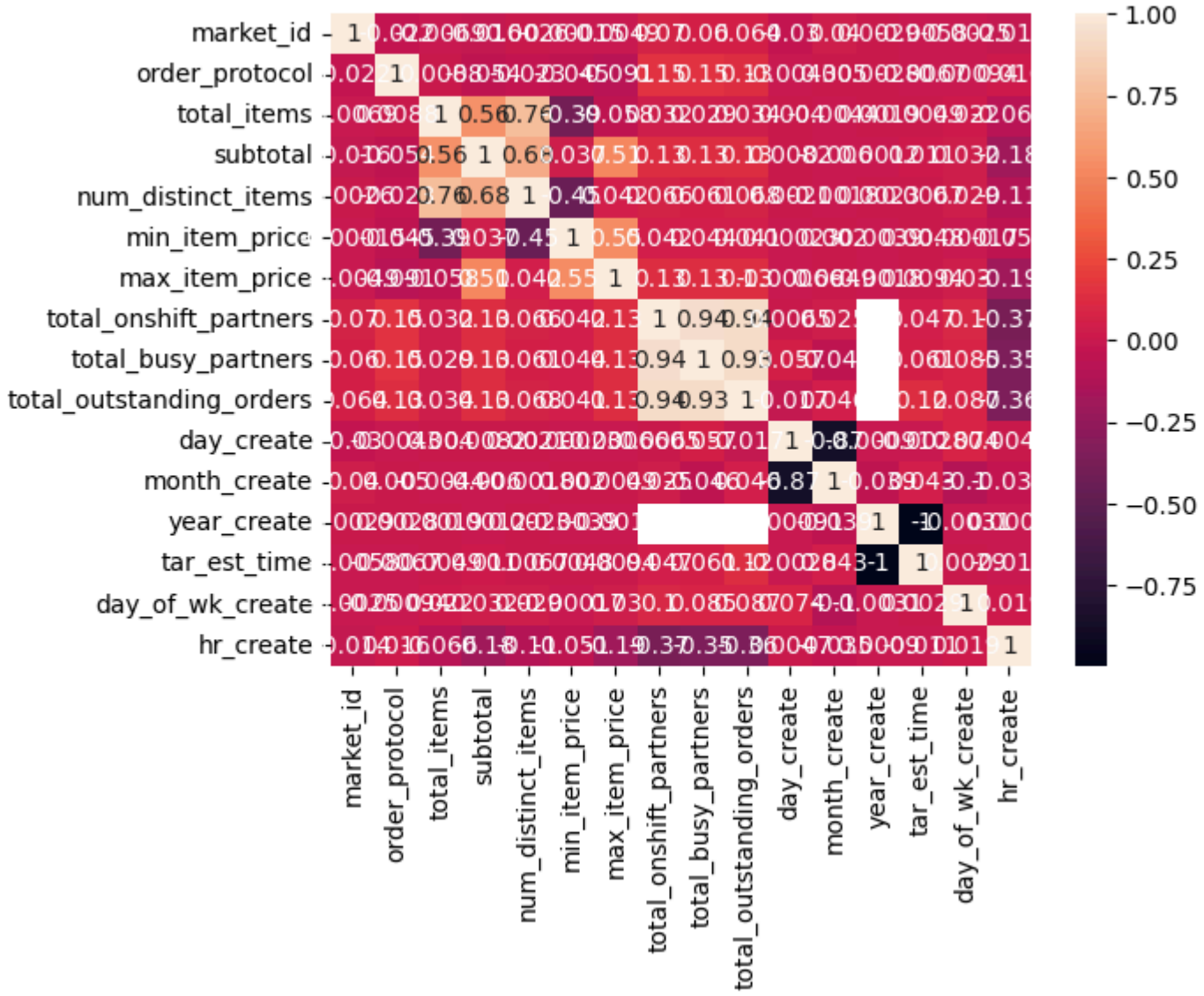
```
cat_cols = ['market_id', 'order_protocol','total_items', 'num_distinct_items', 'subtotal', 'min_item_price', 'max_item_price',
       'total_onshift_partners', 'total_busy_partners',
       'total_outstanding_orders']
```

```
fig, ax = plt.subplots(4,3, figsize = (12,10))

sns.boxplot(ax = ax[0,0], y = df[cat_cols[0]], x = df['tar_est_time'])
sns.boxplot(ax = ax[0,1], y = df[cat_cols[1]], x = df['tar_est_time'])
sns.scatterplot(ax = ax[0,2], y = df[cat_cols[2]], x = df['tar_est_time'])
sns.scatterplot(ax = ax[1,0], y = df[cat_cols[3]], x = df['tar_est_time'])
sns.scatterplot(ax = ax[1,1], y = df[cat_cols[4]], x = df['tar_est_time'])
sns.scatterplot(ax = ax[1,2], y = df[cat_cols[5]], x = df['tar_est_time'])
sns.scatterplot(ax = ax[2,0], y = df[cat_cols[6]], x = df['tar_est_time'])
sns.scatterplot(ax = ax[2,1], y = df[cat_cols[7]], x = df['tar_est_time'])
sns.scatterplot(ax = ax[2,2], y = df[cat_cols[8]], x = df['tar_est_time'])
sns.scatterplot(ax = ax[3,0], y = df[cat_cols[9]], x = df['tar_est_time'])
fig.delaxes(ax[3, 1])
fig.delaxes(ax[3, 2])
plt.tight_layout()
plt.show()
```




```
# removed date and columns with text
df1 = df.drop(columns = ['created_at', 'actual_delivery_time', 'store_id','store_primary_category'])
sns.heatmap(df1.corr(method='pearson'), annot=True)
```

<Axes: >



Data preprocessing

- Checking duplicates
- Handling Missing Data
- Removing the outliers
- Standardizing Data Formats
- Feature engineering

observation found

- No duplicates are available
- outlier are removed in above code
- Missing values are imputed as below using median, mean and KNN methods
- Updated columns category data to numerical data by target encoding

```
# removing duplicates
df1 = df.drop_duplicates()
df1.shape
```

```
(197425, 20)
```

Handling missing data

```
# impute missing values
df['market_id'].fillna(value = df['market_id'].mode()[0],  inplace=True)
df['tar_est_time'].fillna(value = df['tar_est_time'].mean(),  inplace=True)
df['order_protocol'].fillna(value = df['order_protocol'].mode()[0],  inplace=True)
```

```
<ipython-input-32-1c49bac09702>:2: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are sett

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df


  df['market_id'].fillna(value = df['market_id'].mode()[0],  inplace=True)
<ipython-input-32-1c49bac09702>:3: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are sett
```

```
For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df

  df['tar_est_time'].fillna(value = df['tar_est_time'].mean(),  inplace=True)
<ipython-input-32-1c49bac09702>:4: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setti

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df

  df['order_protocol'].fillna(value = df['order_protocol'].mode()[0],  inplace=True)
```

```
# using KNN inputation for below columns
knn_imputer = KNNImputer(n_neighbors=3)

# Apply KNN Imputation
df_imputed = knn_imputer.fit_transform(df[['total_onshift_partners','total_busy_partners', 'total_outstanding_orders']])
df_imputed
```

```
array([[33., 14., 21.],
       [ 4.,  3.,  3.],
       [ 4.,  3.,  3.],
       ...,
       [39., 41., 40.],
       [ 7.,  7., 12.],
       [20., 20., 23.]])
```

```
df_imputed = pd.DataFrame(df_imputed, columns=['total_onshift_partners','total_busy_partners', 'total_outstanding_orders'])
df['total_onshift_partners'] = df_imputed['total_onshift_partners']
df['total_busy_partners'] = df_imputed['total_busy_partners']
df['total_outstanding_orders'] = df_imputed['total_outstanding_orders']
```

```
# handling missing values in store_primary_category column
df['store_primary_category'].fillna(value = df['store_primary_category'].mode()[0],  inplace=True)
```

```
<ipython-input-35-f3e7e4ad6680>:2: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setti

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df

  df['store_primary_category'].fillna(value = df['store_primary_category'].mode()[0],  inplace=True)
```

Target encodeing of store_primary_category column

```
# using target encoding
target_means = df.groupby('store_primary_category')['tar_est_time'].mean()
target_means
```

```
# Impute missing values in 'Feature' using the target encoding
df['store_primary_category'] = df['store_primary_category'].map(target_means)
df['store_primary_category'] = df['store_primary_category'].astype('int64')
```

```
<ipython-input-36-b37aa3f2b64a>:2: FutureWarning: The default of observed=False is deprecated and will be changed to True in a f
  target_means = df.groupby('store_primary_category')['tar_est_time'].mean()
```

Creating X and y data for training / Val / Test

```
X = df.drop(columns = ['created_at', 'actual_delivery_time', 'store_id', 'tar_est_time'])
y = df['tar_est_time']
```

```
X.shape, y.shape
```

```
((197428, 16), (197428,))
```

create Train, val and test data

```
from sklearn.model_selection import train_test_split

X_train_val, X_test, y_train_val, y_test = train_test_split(X, y, test_size=0.1, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val, test_size=0.1, random_state=42)

print('Train : ', X_train.shape, y_train.shape)
print('Validation:', X_val.shape, y_val.shape)
print('Test  : ', X_test.shape, y_test.shape)
```

```
Train :  (159916, 16) (159916,)
Validation: (17769, 16) (17769,)
Test  :  (19743, 16) (19743,)
```

Scaling data for training

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)

X_val = scaler.transform(X_val)
X_test = scaler.transform(X_test)
```

training data

Iteration 1

```
from tensorflow.keras.layers import Dense, Input
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.optimizers import SGD

import tensorflow as tf
from tensorflow.keras.layers import Input, Dense, Conv2D, Flatten, concatenate
from tensorflow.keras.models import Model
```

```
# define varialb as sequential class
model = Sequential()

model.add(Dense(32, input_dim = 16, activation="relu",kernel_initializer='glorot_uniform')) # 1st hidden layer
model.add(Dense(64, activation="relu",kernel_initializer='glorot_uniform')) # 2nd hidden layer
model.add(Dense(128, activation="relu",kernel_initializer='glorot_uniform')) # 3rd hidden layer
model.add(Dense(64, activation="relu",kernel_initializer='glorot_uniform')) # 4th hidden layer
model.add(Dense(32, activation="relu",kernel_initializer='glorot_uniform')) # 5th hidden layer
model.add(Dense(1)) # output layer by default activation function will be linear activation function

model.compile(loss='mean_squared_error', optimizer='adam', metrics=['mean_squared_error'])
```

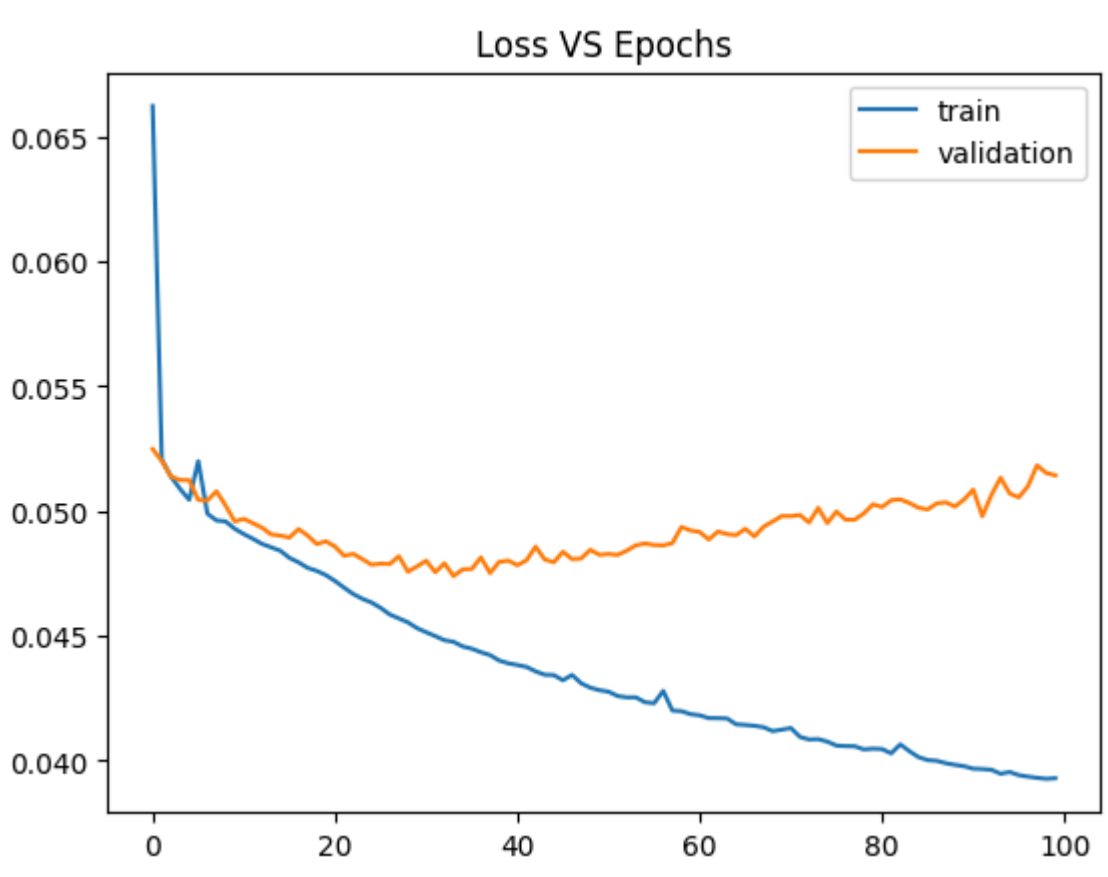
```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim`
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint

early_stopping = EarlyStopping(monitor='mean_squared_error', patience = 3, mode='min')
checkpoint = ModelCheckpoint(filepath='model.h5', save_best_only=True, verbose = 1, monitor='mean_squared_error', mode='min')
```

```
history = model.fit(X_train, y_train, epochs=100, batch_size=256, validation_data = (X_val, y_val), callbacks=[early_stopping, check
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file form
625/625 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - loss: 0.0392 - mean_squared_error: 0.0392 - val_loss: 0.0509 - val_mean_squared_er
Epoch 92/100
618/625 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0392 - mean_squared_error: 0.0392
Epoch 92: mean_squared_error improved from 0.03968 to 0.03967, saving model to model.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file form
625/625 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - loss: 0.0392 - mean_squared_error: 0.0392 - val_loss: 0.0498 - val_mean_squared_er
Epoch 93/100
618/625 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0393 - mean_squared_error: 0.0393
Epoch 93: mean_squared_error improved from 0.03967 to 0.03964, saving model to model.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file form
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - loss: 0.0393 - mean_squared_error: 0.0393 - val_loss: 0.0506 - val_mean_squared_er
Epoch 94/100
619/625 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0390 - mean_squared_error: 0.0390
Epoch 94: mean_squared_error improved from 0.03964 to 0.03948, saving model to model.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file form
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.0390 - mean_squared_error: 0.0390 - val_loss: 0.0513 - val_mean_squared_er
Epoch 95/100
609/625 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0392 - mean_squared_error: 0.0392
Epoch 95: mean_squared_error did not improve from 0.03948
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.0392 - mean_squared_error: 0.0392 - val_loss: 0.0507 - val_mean_squared_er
Epoch 96/100
615/625 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0388 - mean_squared_error: 0.0388
Epoch 96: mean_squared_error improved from 0.03948 to 0.03942, saving model to model.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file form
625/625 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - loss: 0.0388 - mean_squared_error: 0.0388 - val_loss: 0.0505 - val_mean_squared_er
Epoch 97/100
616/625 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0388 - mean_squared_error: 0.0388
Epoch 97: mean_squared_error improved from 0.03942 to 0.03936, saving model to model.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file form
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.0388 - mean_squared_error: 0.0388 - val_loss: 0.0510 - val_mean_squared_er
Epoch 98/100
623/625 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0388 - mean_squared_error: 0.0388
Epoch 98: mean_squared_error improved from 0.03936 to 0.03932, saving model to model.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file form
625/625 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 0.0388 - mean_squared_error: 0.0388 - val_loss: 0.0518 - val_mean_squared_er
Epoch 99/100
619/625 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0391 - mean_squared_error: 0.0391
Epoch 99: mean_squared_error improved from 0.03932 to 0.03928, saving model to model.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file form
625/625 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - loss: 0.0391 - mean_squared_error: 0.0391 - val_loss: 0.0515 - val_mean_squared_er
Epoch 100/100
624/625 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0392 - mean_squared_error: 0.0392
Epoch 100: mean_squared_error did not improve from 0.03928
625/625 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - loss: 0.0392 - mean_squared_error: 0.0392 - val_loss: 0.0514 - val_mean_squared_er
```

```
epochs = history.epoch
loss = history.history["loss"]
val_loss = history.history["val_loss"]


plt.plot(epochs, loss, label="train")
plt.plot(epochs, val_loss, label="validation")

plt.legend()
plt.title("Loss VS Epochs")

plt.show()
```



```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 32)	544
dense_1 (Dense)	(None, 64)	2,112
dense_2 (Dense)	(None, 128)	8,320
dense_3 (Dense)	(None, 64)	8,256
dense_4 (Dense)	(None, 32)	2,080
dense_5 (Dense)	(None, 1)	33

Total params: 64,037 (250.15 KB)
Trainable params: 21,345 (83.38 KB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 42,692 (166.77 KB)

```
from tensorflow.keras.models import load_model
best_model = load_model('model.h5')

pred = best_model.predict(X_test)

from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
import numpy as np

r2_score = r2_score(y_test, pred)
mse = mean_squared_error(y_test, pred)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test, pred)
print("Test acc r2_score: ", r2_score)
print("Test acc mse: ", mse)
print("Test acc rmse: ", rmse)
print("Test acc mae: ", mae)
```

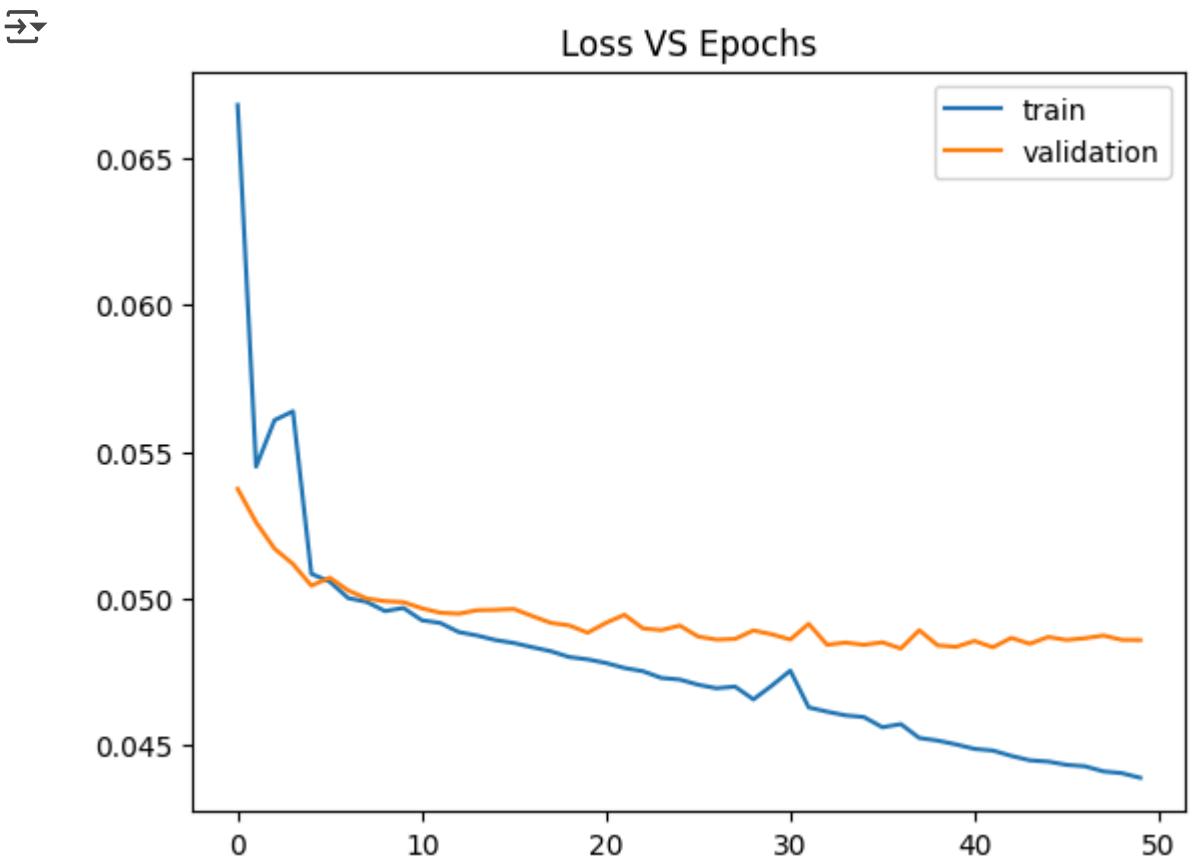
```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty unt
617/617 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step
Test acc r2_score:  0.21097258288955967
Test acc mse:  0.051249353985376
Test acc rmse:  0.2263832016413232
Test acc mae:  0.18051552454158223
```

Iteration 2

```
# define varialb as sequential class
model2 = Sequential()

model2.add(Dense(32, input_dim = 16, activation="relu")) # 1st hidden layer
model2.add(Dense(64, activation="relu")) # 2nd hidden layer
model2.add(Dense(128, activation="relu")) # 3rd hidden layer
model2.add(Dense(64, activation="relu")) # 4th hidden layer
model2.add(Dense(32, activation="relu")) # 5th hidden layer
model2.add(Dense(1)) # output layer by default activation function will be linear activation function

model2.compile(loss='mean_squared_error', optimizer='adam', metrics=['mse','r2_score'])

early_stopping_2 = EarlyStopping(monitor='r2_score', patience = 3, mode='max')
checkpoint_2 = ModelCheckpoint(filepath='model_2.h5', save_best_only=True, verbose = 1, monitor='r2_score', mode='max')

history_2 = model2.fit(X_train, y_train, epochs=50, batch_size=256, validation_data = (X_val, y_val), callbacks=[early_stopping_2, c
```

```
613/625 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0444 - mse: 0.0444 - r2_score: 0.3251
Epoch 42: r2_score improved from 0.31959 to 0.32045, saving model to model_2.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file form
625/625 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - loss: 0.0444 - mse: 0.0444 - r2_score: 0.3250 - val_loss: 0.0483 - val_mse: 0.0483
Epoch 43/50
613/625 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0442 - mse: 0.0442 - r2_score: 0.3257
Epoch 43: r2_score improved from 0.32045 to 0.32321, saving model to model_2.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file form
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.0442 - mse: 0.0442 - r2_score: 0.3256 - val_loss: 0.0486 - val_mse: 0.0486
Epoch 44/50
621/625 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0443 - mse: 0.0443 - r2_score: 0.3264
Epoch 44: r2_score improved from 0.32321 to 0.32550, saving model to model_2.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file form
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - loss: 0.0443 - mse: 0.0443 - r2_score: 0.3264 - val_loss: 0.0484 - val_mse: 0.0484
Epoch 45/50
623/625 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0444 - mse: 0.0444 - r2_score: 0.3266
Epoch 45: r2_score improved from 0.32550 to 0.32615, saving model to model_2.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file form
625/625 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - loss: 0.0444 - mse: 0.0444 - r2_score: 0.3266 - val_loss: 0.0487 - val_mse: 0.0487
Epoch 46/50
621/625 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0443 - mse: 0.0443 - r2_score: 0.3287
Epoch 46: r2_score improved from 0.32615 to 0.32784, saving model to model_2.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file form
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.0443 - mse: 0.0443 - r2_score: 0.3287 - val_loss: 0.0486 - val_mse: 0.0486
Epoch 47/50
622/625 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0443 - mse: 0.0443 - r2_score: 0.3284
Epoch 47: r2_score improved from 0.32784 to 0.32857, saving model to model_2.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file form
625/625 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - loss: 0.0443 - mse: 0.0443 - r2_score: 0.3284 - val_loss: 0.0486 - val_mse: 0.0486
Epoch 48/50
620/625 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0438 - mse: 0.0438 - r2_score: 0.3335
Epoch 48: r2_score improved from 0.32857 to 0.33124, saving model to model_2.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file form
625/625 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 0.0438 - mse: 0.0438 - r2_score: 0.3335 - val_loss: 0.0487 - val_mse: 0.0487
Epoch 49/50
612/625 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0441 - mse: 0.0441 - r2_score: 0.3339
Epoch 49: r2_score improved from 0.33124 to 0.33215, saving model to model_2.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file form
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.0441 - mse: 0.0441 - r2_score: 0.3338 - val_loss: 0.0486 - val_mse: 0.0486
Epoch 50/50
618/625 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0434 - mse: 0.0434 - r2_score: 0.3414
Epoch 50: r2_score improved from 0.33215 to 0.33456, saving model to model_2.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file form
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.0434 - mse: 0.0434 - r2_score: 0.3413 - val_loss: 0.0486 - val_mse: 0.0486
```

```
epochs = history_2.epoch
loss = history_2.history["loss"]
val_loss = history_2.history["val_loss"]


plt.plot(epochs, loss, label="train")
plt.plot(epochs, val_loss, label="validation")

plt.legend()
plt.title("Loss VS Epochs")

plt.show()
```



```
from tensorflow.keras.models import load_model
best_model = load_model('model_2.h5')

pred = best_model.predict(X_test)

from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
import numpy as np

r2_score = r2_score(y_test, pred)
mse = mean_squared_error(y_test, pred)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test, pred)
print("Test acc r2_score: ", r2_score)
print("Test acc mse: ", mse)
print("Test acc rmse: ", rmse)
print("Test acc mae: ", mae)
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty unt
617/617 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step
Test acc r2_score:  0.2686326093250557
Test acc mse:  0.04750418741509285
Test acc rmse:  0.2179545535543886
Test acc mae:  0.1743037704500859
```

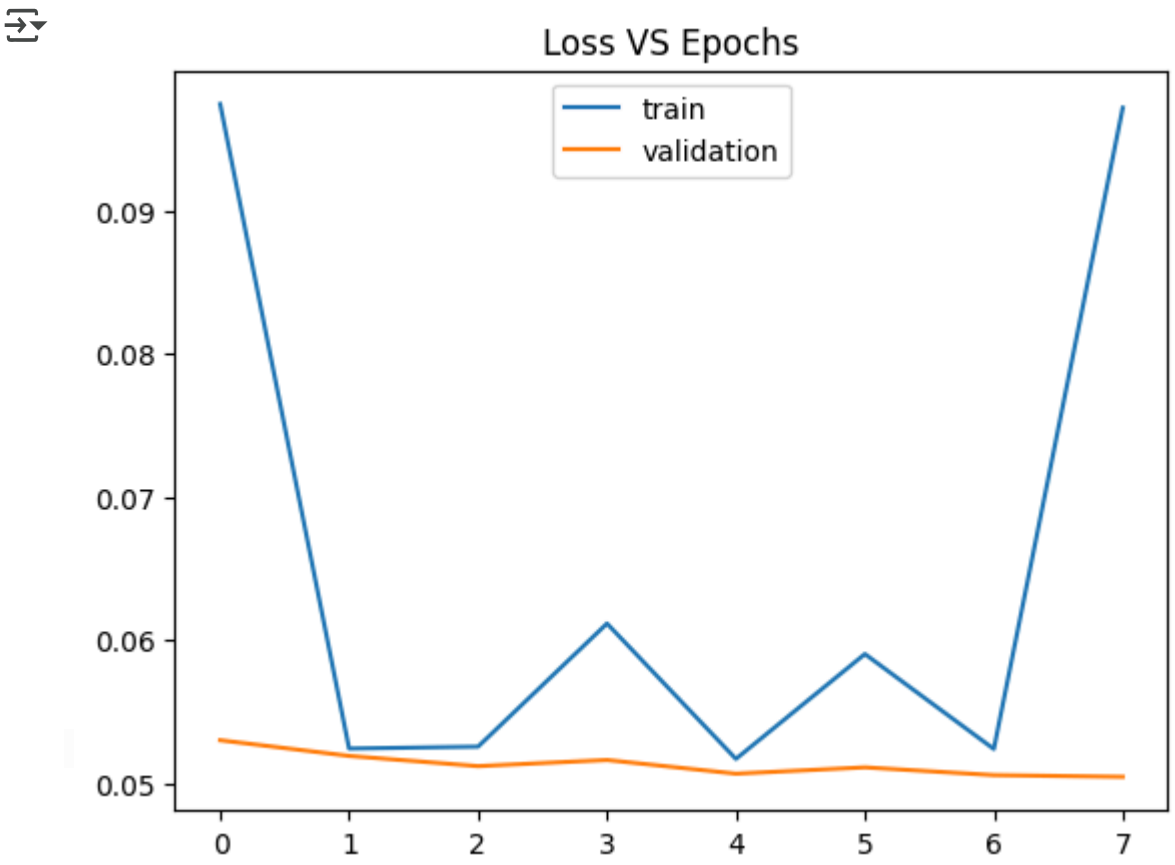
Iteration 3

```
# define varialb as sequential class
model3 = Sequential()

model3.add(Dense(32, input_dim = 16, activation="leaky_relu")) # 1st hidden layer
model3.add(Dense(64, activation="leaky_relu")) # 2nd hidden layer
model3.add(Dense(128, activation="leaky_relu")) # 3rd hidden layer
model3.add(Dense(64, activation="leaky_relu")) # 4th hidden layer
model3.add(Dense(32, activation="leaky_relu")) # 5th hidden layer
model3.add(Dense(1)) # output layer by default activation function will be linear activation function

model3.compile(loss='mean_squared_error', optimizer='adam', metrics=['mse','r2_score'])

early_stopping_3 = EarlyStopping(monitor='r2_score', patience = 3, mode='max')
checkpoint_3 = ModelCheckpoint(filepath='model_3.h5', save_best_only=True, verbose = 1, monitor='r2_score', mode='max')

history_3 = model3.fit(X_train, y_train, epochs=70, batch_size=256, validation_data = (X_val, y_val), callbacks=[early_stopping_3, c
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim`
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/70
621/625 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.1638 - mse: 0.1638 - r2_score: -1.4719
Epoch 1: r2_score improved from -inf to -0.47781, saving model to model_3.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format
625/625 ━━━━━━━━━━━━━━━━━━━━ 10s 7ms/step - loss: 0.1632 - mse: 0.1632 - r2_score: -1.4640 - val_loss: 0.0530 - val_mse: 0.0530
Epoch 2/70
614/625 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0530 - mse: 0.0530 - r2_score: 0.1959
Epoch 2: r2_score improved from -0.47781 to 0.20495, saving model to model_3.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format
625/625 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - loss: 0.0530 - mse: 0.0530 - r2_score: 0.1961 - val_loss: 0.0519 - val_mse: 0.0519 -
Epoch 3/70
621/625 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0531 - mse: 0.0531 - r2_score: 0.1933
Epoch 3: r2_score did not improve from 0.20495
625/625 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - loss: 0.0531 - mse: 0.0531 - r2_score: 0.1934 - val_loss: 0.0512 - val_mse: 0.0512 -
Epoch 4/70
613/625 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0590 - mse: 0.0590 - r2_score: 0.1077
Epoch 4: r2_score did not improve from 0.20495
625/625 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - loss: 0.0590 - mse: 0.0590 - r2_score: 0.1070 - val_loss: 0.0516 - val_mse: 0.0516 -
Epoch 5/70
621/625 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0519 - mse: 0.0519 - r2_score: 0.2118
Epoch 5: r2_score improved from 0.20495 to 0.21593, saving model to model_3.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format
625/625 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - loss: 0.0519 - mse: 0.0519 - r2_score: 0.2119 - val_loss: 0.0507 - val_mse: 0.0507 -
Epoch 6/70
624/625 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0529 - mse: 0.0529 - r2_score: 0.1943
Epoch 6: r2_score did not improve from 0.21593
625/625 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - loss: 0.0529 - mse: 0.0529 - r2_score: 0.1940 - val_loss: 0.0511 - val_mse: 0.0511 -
Epoch 7/70
612/625 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0519 - mse: 0.0519 - r2_score: 0.2111
Epoch 7: r2_score did not improve from 0.21593
625/625 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - loss: 0.0519 - mse: 0.0519 - r2_score: 0.2110 - val_loss: 0.0506 - val_mse: 0.0506 -
Epoch 8/70
623/625 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.0931 - mse: 0.0931 - r2_score: -0.4108
Epoch 8: r2_score did not improve from 0.21593
```

```
625/625 ━━━━━━━━━━━━━━━━━━━━ 5s 8ms/step - loss: 0.0931 - mse: 0.0931 - r2_score: -0.4111 - val_loss: 0.0504 - val_mse: 0.0504 -
```

```
epochs = history_3.epoch
loss = history_3.history["loss"]
val_loss = history_3.history["val_loss"]


plt.plot(epochs, loss, label="train")
plt.plot(epochs, val_loss, label="validation")

plt.legend()
plt.title("Loss VS Epochs")

plt.show()
```




```
from tensorflow.keras.models import load_model
best_model = load_model('model_3.h5')

pred = best_model.predict(X_test)

from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
import numpy as np

r2_score = r2_score(y_test, pred)
mse = mean_squared_error(y_test, pred)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test, pred)
print("Test acc r2_score: ", r2_score)
print("Test acc mse: ", mse)
print("Test acc rmse: ", rmse)
print("Test acc mae: ", mae)
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty unt
617/617 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step
Test acc r2_score:  0.226339651578924
Test acc mse:  0.05025122346937571
Test acc rmse:  0.2241678466448204
Test acc mae:  0.17936568635899341
```

Iteratin 4

```
# define varialb as sequential class
model4 = Sequential()

model4.add(Dense(16, input_dim = 16, activation="leaky_relu")) # 1st hidden layer
model4.add(Dense(32, activation="leaky_relu")) # 2nd hidden layer
model4.add(Dense(64, activation="leaky_relu")) # 3rd hidden layer
model4.add(Dense(128, activation="leaky_relu")) # 4th hidden layer
model4.add(Dense(64, activation="leaky_relu")) # 5th hidden layer
model4.add(Dense(32, activation="leaky_relu")) # 6th hidden layer
```

```
model4.add(Dense(16, activation="leaky_relu")) # 7th hidden layer
model4.add(Dense(1)) # output layer by default activation function will be linear activation function

model4.compile(loss='mse', optimizer='adam', metrics=['r2_score'])

model4.fit(X_train, y_train, epochs=20, batch_size=256, validation_data = (X_val, y_val))
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim`
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 6s 5ms/step - loss: 0.0943 - r2_score: -0.4341 - val_loss: 0.0536 - val_r2_score: 0.1784
Epoch 2/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - loss: 0.0535 - r2_score: 0.1893 - val_loss: 0.0541 - val_r2_score: 0.1707
Epoch 3/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 6s 5ms/step - loss: 0.0554 - r2_score: 0.1558 - val_loss: 0.0515 - val_r2_score: 0.2102
Epoch 4/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 0.0554 - r2_score: 0.1612 - val_loss: 0.0514 - val_r2_score: 0.2109
Epoch 5/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - loss: 0.0697 - r2_score: -0.0547 - val_loss: 0.0513 - val_r2_score: 0.2133
Epoch 6/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - loss: 0.0522 - r2_score: 0.2103 - val_loss: 0.0512 - val_r2_score: 0.2148
Epoch 7/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - loss: 0.0882 - r2_score: -0.3467 - val_loss: 0.0517 - val_r2_score: 0.2067
Epoch 8/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 0.0511 - r2_score: 0.2235 - val_loss: 0.0510 - val_r2_score: 0.2175
Epoch 9/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.0649 - r2_score: 0.0132 - val_loss: 0.0508 - val_r2_score: 0.2212
Epoch 10/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - loss: 0.0505 - r2_score: 0.2335 - val_loss: 0.0509 - val_r2_score: 0.2190
Epoch 11/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 6s 5ms/step - loss: 0.0506 - r2_score: 0.2331 - val_loss: 0.0506 - val_r2_score: 0.2236
Epoch 12/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - loss: 0.0503 - r2_score: 0.2334 - val_loss: 0.0504 - val_r2_score: 0.2265
Epoch 13/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.0504 - r2_score: 0.2366 - val_loss: 0.0509 - val_r2_score: 0.2199
Epoch 14/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 0.0501 - r2_score: 0.2381 - val_loss: 0.0507 - val_r2_score: 0.2226
Epoch 15/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.0501 - r2_score: 0.2385 - val_loss: 0.0506 - val_r2_score: 0.2237
Epoch 16/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - loss: 0.0500 - r2_score: 0.2415 - val_loss: 0.0503 - val_r2_score: 0.2285
Epoch 17/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - loss: 0.0505 - r2_score: 0.2393 - val_loss: 0.0501 - val_r2_score: 0.2316
Epoch 18/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.0500 - r2_score: 0.2435 - val_loss: 0.0501 - val_r2_score: 0.2309
Epoch 19/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - loss: 0.0528 - r2_score: 0.2040 - val_loss: 0.0510 - val_r2_score: 0.2181
Epoch 20/20
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.0499 - r2_score: 0.2460 - val_loss: 0.0504 - val_r2_score: 0.2263
<keras.src.callbacks.history.History at 0x7f2246dd4b90>
```

```
pred = model4.predict(X_test)

from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
import numpy as np

r2_score = r2_score(y_test, pred)
mse = mean_squared_error(y_test, pred)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test, pred)
print("Test acc r2_score: ", r2_score)
print("Test acc mse: ", mse)
print("Test acc rmse: ", rmse)
print("Test acc mae: ", mae)
```

```
617/617 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
Test acc r2_score:  0.23361439463565903
Test acc mse:  0.04977871025376068
Test acc rmse:  0.22311143012799833
Test acc mae:  0.1750757555013447
```

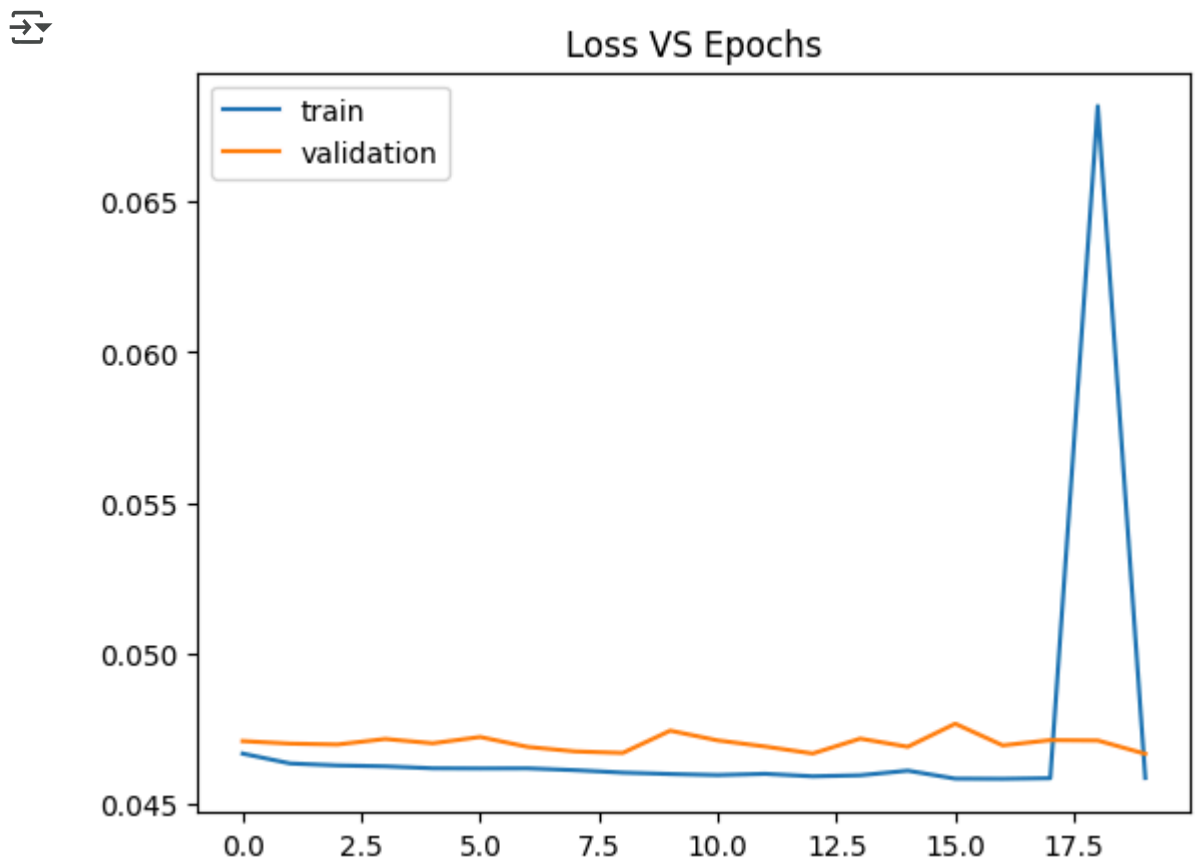
Iteration 5

```
# define varialb as sequential class
from tensorflow.keras.layers import Dense, Dropout
model5 = Sequential()

model5.add(Dense(32, input_dim = 16, activation="tanh")) # 1st hidden layer
Dropout(0.5)
```

```
model5.add(Dense(64, activation="leaky_relu")) # 2nd hidden layer
Dropout(0.3)
model5.add(Dense(32, activation="leaky_relu")) # 3rd hidden layer
Dropout(0.2)
model5.add(Dense(1)) # output layer by default activation function will be linear activation function

model5.compile(loss='mean_squared_error', optimizer='adam', metrics=['mse','r2_score'])

early_stopping_5 = EarlyStopping(monitor='r2_score', patience = 3, mode='max')
checkpoint_5 = ModelCheckpoint(filepath='model_5.h5', save_best_only=True, verbose = 1, monitor='r2_score', mode='max')

history_5 = model3.fit(X_train, y_train, epochs=70, batch_size=256, validation_data = (X_val, y_val), callbacks=[early_stopping_5, c
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_d
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/70
617/625 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.0473 - mse: 0.0473 - r2_score: 0.2840
Epoch 1: r2_score improved from -inf to 0.29227, saving model to model_5.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file form
625/625 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - loss: 0.0473 - mse: 0.0473 - r2_score: 0.2841 - val_loss: 0.0471 - val_mse: 0.0471
Epoch 2/70
624/625 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.0460 - mse: 0.0460 - r2_score: 0.2989
Epoch 2: r2_score improved from 0.29227 to 0.29724, saving model to model_5.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file form
625/625 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 0.0460 - mse: 0.0460 - r2_score: 0.2989 - val_loss: 0.0470 - val_mse: 0.0470
Epoch 3/70
621/625 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0460 - mse: 0.0460 - r2_score: 0.3013
Epoch 3: r2_score improved from 0.29724 to 0.29824, saving model to model_5.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file form
625/625 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - loss: 0.0460 - mse: 0.0460 - r2_score: 0.3013 - val_loss: 0.0470 - val_mse: 0.0470
Epoch 4/70
616/625 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0463 - mse: 0.0463 - r2_score: 0.3031
Epoch 4: r2_score improved from 0.29824 to 0.29868, saving model to model_5.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file form
625/625 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - loss: 0.0463 - mse: 0.0463 - r2_score: 0.3030 - val_loss: 0.0471 - val_mse: 0.0471
Epoch 5/70
620/625 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0462 - mse: 0.0462 - r2_score: 0.2990
Epoch 5: r2_score improved from 0.29868 to 0.29967, saving model to model_5.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file form
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.0462 - mse: 0.0462 - r2_score: 0.2990 - val_loss: 0.0470 - val_mse: 0.0470
Epoch 6/70
619/625 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0460 - mse: 0.0460 - r2_score: 0.3002
Epoch 6: r2_score improved from 0.29967 to 0.29976, saving model to model_5.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file form
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - loss: 0.0460 - mse: 0.0460 - r2_score: 0.3002 - val_loss: 0.0472 - val_mse: 0.0472
Epoch 7/70
616/625 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0462 - mse: 0.0462 - r2_score: 0.2975
Epoch 7: r2_score did not improve from 0.29976
625/625 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - loss: 0.0462 - mse: 0.0462 - r2_score: 0.2975 - val_loss: 0.0469 - val_mse: 0.0469
Epoch 8/70
625/625 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0458 - mse: 0.0458 - r2_score: 0.3040
Epoch 8: r2_score improved from 0.29976 to 0.30062, saving model to model_5.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file form
625/625 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - loss: 0.0458 - mse: 0.0458 - r2_score: 0.3040 - val_loss: 0.0467 - val_mse: 0.0467
Epoch 9/70
619/625 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0460 - mse: 0.0460 - r2_score: 0.3028
Epoch 9: r2_score improved from 0.30062 to 0.30185, saving model to model_5.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file form
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.0460 - mse: 0.0460 - r2_score: 0.3028 - val_loss: 0.0467 - val_mse: 0.0467
Epoch 10/70
614/625 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0460 - mse: 0.0460 - r2_score: 0.3023
Epoch 10: r2_score improved from 0.30185 to 0.30256, saving model to model_5.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file form
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.0460 - mse: 0.0460 - r2_score: 0.3023 - val_loss: 0.0474 - val_mse: 0.0474
Epoch 11/70
617/625 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0459 - mse: 0.0459 - r2_score: 0.3049
Epoch 11: r2_score improved from 0.30256 to 0.30310, saving model to model_5.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file form
625/625 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 0.0459 - mse: 0.0459 - r2_score: 0.3049 - val_loss: 0.0471 - val_mse: 0.0471
Epoch 12/70
```

```
epochs = history_5.epoch
loss = history_5.history["loss"]
val_loss = history_5.history["val_loss"]


plt.plot(epochs, loss, label="train")
plt.plot(epochs, val_loss, label="validation")

plt.legend()
plt.title("Loss VS Epochs")

plt.show()
```




```
from tensorflow.keras.models import load_model
best_model = load_model('model_5.h5')

pred = best_model.predict(X_test)

from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
import numpy as np

r2_score = r2_score(y_test, pred)
mse = mean_squared_error(y_test, pred)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test, pred)
print("Test acc r2_score: ", r2_score)
print("Test acc mse: ", mse)
print("Test acc rmse: ", rmse)
print("Test acc mae: ", mae)
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty unt
617/617 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step
Test acc r2_score:  0.28944548667078973
Test acc mse:  0.04615233766258111
Test acc rmse:  0.21483095136078764
Test acc mae:  0.16977076446846945
```

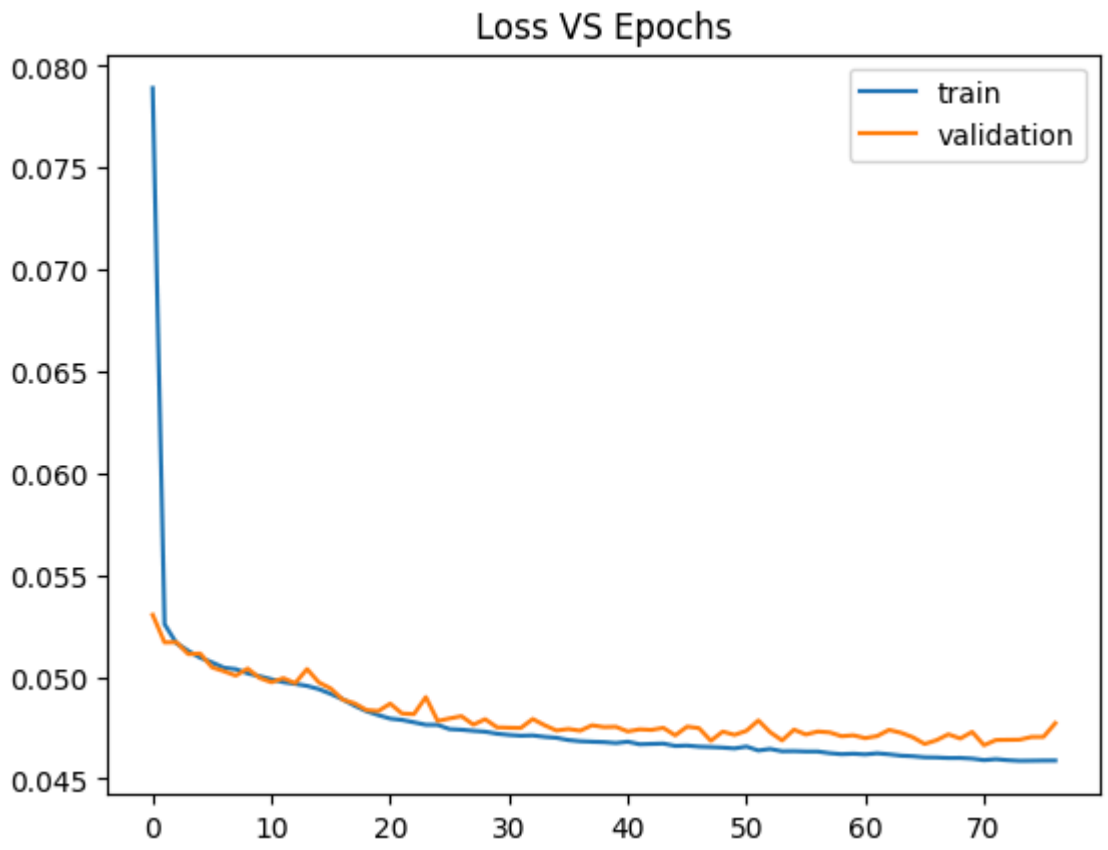
Iteration 6

```
# define varialb as sequential class
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization
model6 = Sequential()

model6.add(Dense(32, input_dim = 16, activation="tanh")) # 1st hidden layer
BatchNormalization()
Dropout(0.5)
model6.add(Dense(64, activation="leaky_relu")) # 2nd hidden layer
BatchNormalization()
Dropout(0.3)
model6.add(Dense(32, activation="leaky_relu")) # 3rd hidden layer
BatchNormalization()
Dropout(0.2)
```

```
model6.add(Dense(1)) # output layer by default activation function will be linear activation function

model6.compile(loss='mean_squared_error', optimizer='adam', metrics=['mse','r2_score'])

early_stopping_6 = EarlyStopping(monitor='r2_score', patience = 3, mode='max')
checkpoint_6 = ModelCheckpoint(filepath='model_6.h5', save_best_only=True, verbose = 1, monitor='r2_score', mode='max')

history_6 = model6.fit(X_train, y_train, epochs=100, batch_size=256, validation_data = (X_val, y_val), callbacks=[early_stopping_6,
```

```
Epoch 1/100
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_d
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
612/625 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.1675 - mse: 0.1675 - r2_score: -1.5808
Epoch 1: r2_score improved from -inf to -0.19700, saving model to model_6.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file form
625/625 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - loss: 0.1655 - mse: 0.1655 - r2_score: -1.5500 - val_loss: 0.0531 - val_mse: 0.053
Epoch 2/100
621/625 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0528 - mse: 0.0528 - r2_score: 0.1956
Epoch 2: r2_score improved from -0.19700 to 0.20223, saving model to model_6.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file form
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - loss: 0.0528 - mse: 0.0528 - r2_score: 0.1957 - val_loss: 0.0517 - val_mse: 0.0517
Epoch 3/100
617/625 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0520 - mse: 0.0520 - r2_score: 0.2139
Epoch 3: r2_score improved from 0.20223 to 0.21641, saving model to model_6.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file form
625/625 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - loss: 0.0520 - mse: 0.0520 - r2_score: 0.2139 - val_loss: 0.0517 - val_mse: 0.0517
Epoch 4/100
617/625 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.0510 - mse: 0.0510 - r2_score: 0.2245
Epoch 4: r2_score improved from 0.21641 to 0.22219, saving model to model_6.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file form
625/625 ━━━━━━━━━━━━━━━━━━━━ 7s 8ms/step - loss: 0.0511 - mse: 0.0511 - r2_score: 0.2244 - val_loss: 0.0511 - val_mse: 0.0511
Epoch 5/100
625/625 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0509 - mse: 0.0509 - r2_score: 0.2285
Epoch 5: r2_score improved from 0.22219 to 0.22719, saving model to model_6.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file form
625/625 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - loss: 0.0509 - mse: 0.0509 - r2_score: 0.2285 - val_loss: 0.0512 - val_mse: 0.0512
Epoch 6/100
621/625 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.0505 - mse: 0.0505 - r2_score: 0.2343
Epoch 6: r2_score improved from 0.22719 to 0.23075, saving model to model_6.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file form
625/625 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - loss: 0.0505 - mse: 0.0505 - r2_score: 0.2343 - val_loss: 0.0505 - val_mse: 0.0505
Epoch 7/100
617/625 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0506 - mse: 0.0506 - r2_score: 0.2328
Epoch 7: r2_score improved from 0.23075 to 0.23471, saving model to model_6.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file form
625/625 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 0.0506 - mse: 0.0506 - r2_score: 0.2328 - val_loss: 0.0503 - val_mse: 0.0503
Epoch 8/100
622/625 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0502 - mse: 0.0502 - r2_score: 0.2371
Epoch 8: r2_score improved from 0.23471 to 0.23572, saving model to model_6.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file form
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - loss: 0.0502 - mse: 0.0502 - r2_score: 0.2371 - val_loss: 0.0501 - val_mse: 0.0501
Epoch 9/100
620/625 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0504 - mse: 0.0504 - r2_score: 0.2375
Epoch 9: r2_score improved from 0.23572 to 0.23889, saving model to model_6.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file form
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - loss: 0.0504 - mse: 0.0504 - r2_score: 0.2375 - val_loss: 0.0504 - val_mse: 0.0504
Epoch 10/100
600/625 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0502 - mse: 0.0502 - r2_score: 0.2386
Epoch 10: r2_score improved from 0.23889 to 0.24106, saving model to model_6.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file form
625/625 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - loss: 0.0502 - mse: 0.0502 - r2_score: 0.2387 - val_loss: 0.0500 - val_mse: 0.0500
Epoch 11/100
605/625 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.0499 - mse: 0.0499 - r2_score: 0.2416
Epoch 11: r2_score improved from 0.24106 to 0.24331, saving model to model_6.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file form
625/625 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 0.0499 - mse: 0.0499 - r2_score: 0.2417 - val_loss: 0.0497 - val_mse: 0.0497
```

```
epochs = history_6.epoch
loss = history_6.history["loss"]
val_loss = history_6.history["val_loss"]


plt.plot(epochs, loss, label="train")
plt.plot(epochs, val_loss, label="validation")

plt.legend()
plt.title("Loss VS Epochs")

plt.show()
```




```
from tensorflow.keras.models import load_model
best_model = load_model('model_6.h5')

pred = best_model.predict(X_test)

from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
import numpy as np

r2_score = r2_score(y_test, pred)
mse = mean_squared_error(y_test, pred)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test, pred)
print("Test acc r2_score: ", r2_score)
print("Test acc mse: ", mse)
print("Test acc rmse: ", rmse)
print("Test acc mae: ", mae)
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty unt
617/617 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step
Test acc r2_score:  0.2862416394971815
Test acc mse:  0.046360435751892894
Test acc rmse:  0.2153147364949573
Test acc mae:  0.17155365738282108
```

Iternation 7

- l1/l2 regularization

```
def create_baseline():
    # lambda = 0.01
    L2Reg = tf.keras.regularizers.L2(l2=1e-6)
    model = Sequential([
                    Dense(256, activation="relu", kernel_regularizer = L2Reg ),
                    Dense(128, activation="relu", kernel_regularizer = L2Reg),
                    Dense(64, activation="relu", kernel_regularizer = L2Reg),
                    Dense(1 , activation = 'sigmoid')])
    return model


model_7 = create_baseline()

model_7.compile(optimizer = tf.keras.optimizers.Adam(),
                loss = tf.keras.losses.BinaryCrossentropy(),
                metrics=["accuracy"])


history = model_7.fit(X_train, y_train, validation_data = (X_val, y_val),  epochs=30, batch_size=128, verbose=1)
```

```
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 1.3933e-04 - loss: -2344.5378 - val_accuracy: 1.6883e-04 - val_loss: -
Epoch 4/30
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 1.6003e-04 - loss: -11008.8604 - val_accuracy: 2.2511e-04 - val_loss:
Epoch 5/30
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 1.6434e-04 - loss: -32389.1484 - val_accuracy: 2.2511e-04 - val_loss:
Epoch 6/30
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 1.1630e-04 - loss: -68516.1875 - val_accuracy: 1.6883e-04 - val_loss:
Epoch 7/30
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 1.4429e-04 - loss: -124107.8438 - val_accuracy: 1.6883e-04 - val_loss:
Epoch 8/30
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 1.3274e-04 - loss: -195963.9844 - val_accuracy: 2.2511e-04 - val_loss:
Epoch 9/30
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 1.1191e-04 - loss: -314694.0938 - val_accuracy: 1.6883e-04 - val_loss:
Epoch 10/30
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 1.2134e-04 - loss: -437363.1250 - val_accuracy: 2.2511e-04 - val_loss:
Epoch 11/30
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 1.8027e-04 - loss: -661060.4375 - val_accuracy: 2.2511e-04 - val_loss:
Epoch 12/30
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 1.5629e-04 - loss: -829921.1875 - val_accuracy: 2.2511e-04 - val_loss:
Epoch 13/30
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 9.8142e-05 - loss: -1295436.5000 - val_accuracy: 1.6883e-04 - val_loss
Epoch 14/30
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 1.3232e-04 - loss: -1690714.3750 - val_accuracy: 2.2511e-04 - val_loss
Epoch 15/30
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 1.3082e-04 - loss: -2137975.5000 - val_accuracy: 1.6883e-04 - val_loss
Epoch 16/30
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 1.5261e-04 - loss: -2546050.0000 - val_accuracy: 2.2511e-04 - val_loss
Epoch 17/30
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 1.4260e-04 - loss: -3185785.2500 - val_accuracy: 2.2511e-04 - val_loss
Epoch 18/30
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 9s 3ms/step - accuracy: 9.2853e-05 - loss: -3823176.7500 - val_accuracy: 2.2511e-04 - val_loss
Epoch 19/30
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 1.4846e-04 - loss: -4680461.5000 - val_accuracy: 2.2511e-04 - val_loss
Epoch 20/30
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 1.3369e-04 - loss: -5555139.5000 - val_accuracy: 2.2511e-04 - val_loss
Epoch 21/30
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 1.5450e-04 - loss: -6963439.5000 - val_accuracy: 1.6883e-04 - val_loss
Epoch 22/30
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 9s 3ms/step - accuracy: 1.0070e-04 - loss: -8518932.0000 - val_accuracy: 2.2511e-04 - val_loss
Epoch 23/30
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 1.4460e-04 - loss: -9459037.0000 - val_accuracy: 2.2511e-04 - val_loss
Epoch 24/30
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - accuracy: 1.3623e-04 - loss: -10997285.0000 - val_accuracy: 2.2511e-04 - val_lo
Epoch 25/30
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 7.8931e-05 - loss: -13394108.0000 - val_accuracy: 1.6883e-04 - val_los
Epoch 26/30
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 9s 7ms/step - accuracy: 1.7171e-04 - loss: -15499116.0000 - val_accuracy: 2.2511e-04 - val_los
Epoch 27/30
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 1.4678e-04 - loss: -16806434.0000 - val_accuracy: 1.6883e-04 - val_los
Epoch 28/30
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 10s 8ms/step - accuracy: 1.6137e-04 - loss: -18587156.0000 - val_accuracy: 1.6883e-04 - val_lo
Epoch 29/30
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 1.6582e-04 - loss: -22755326.0000 - val_accuracy: 2.2511e-04 - val_los
Epoch 30/30
```

```
pred = model_7.predict(X_test)
```

```
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
import numpy as np

r2_score = r2_score(y_test, pred)
mse = mean_squared_error(y_test, pred)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test, pred)
print("Test acc r2_score: ", r2_score)
print("Test acc mse: ", mse)
print("Test acc rmse: ", rmse)
print("Test acc mae: ", mae)
```

```
617/617 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step
Test acc r2_score:  -0.7330571163783237
Test acc mse:  0.11256650365764893
Test acc rmse:  0.33550931977763143
Test acc mae:  0.2929052350481465
```

Summary of all iteration

R2_score are mentioned below

- Iter1 --> 0.210 (with kernal intializers and 5 hidden layers)

- Iter2 --> 0.268 (with only 5 hidden layers)
- Iter3--> 0.226 (with leaky relu activation function and 5 hidden layers)
- Iter4 --> 0.23 (with leaky relu and 7 hidden layer)
- Iter5 --> 0.289 (with drop out with 3 hidden layers)
- Iter6 --> 0.286 (with dropout and batch normalization with 3 hidden layers)
- Iter7 --> -0.733 (with regularization)

Recommendation - Best option are having dropout and batch normalization with more hidden layers - best model

Leading Questions:

1. Defining the problem statements and where can this and modifications of this be used?