```
import numpy as np

import pandas as pd

import plotly.express as px

import matplotlib.pyplot as plt

import seaborn as sns


from sklearn.preprocessing import OneHotEncoder, StandardScaler

from sklearn.model_selection import train_test_split

from sklearn import metrics

from sklearn.pipeline import make_pipeline

from sklearn.ensemble import RandomForestRegressor


import warnings

warnings.filterwarnings('ignore')


# Importing Raw Files

train_raw = pd.read_csv('train.csv')

test_raw = pd.read_csv('test.csv')

meal = pd.read_csv('meal_info.csv')

centerinfo = pd.read_csv('fulfilment_center_info.csv')
```

```
# Analyzing Data

print("The Shape of Demand dataset :", train_raw.shape)

print("The Shape of Fulfillment Center Information dataset :", centerinfo.shape)

print("The Shape of Meal information dataset :", meal.shape)

print("The Shape of Test dataset :", test_raw.shape)

train_raw.head()

centerinfo.head()

meal.head()

test_raw.head()

# Check for missing values

train_raw.isnull().sum().sum()

test_raw.isnull().sum().sum()

# Analysis report

print("The company  has", centerinfo["center_id"].nunique(), " warehouse ", "spreed
into  ",

      centerinfo["city_code"].nunique(), "City and ",
centerinfo["region_code"].nunique(), "Regions")

print("The products of the company are ", meal["meal_id"].nunique(), "unique meals
, divided into  ",

      meal["category"].nunique(), "category and ", meal["cuisine"].nunique(),
"cuisine")

# Merge meal,center-info data with train and test data

train = pd.merge(train_raw, meal, on="meal_id", how="left")

train = pd.merge(train, centerinfo, on="center_id", how="left")

print("Shape of train data : ", train.shape)

train.head()

# Merge test data with meal and center info

test = pd.merge(test_raw, meal, on="meal_id", how="outer")

test = pd.merge(test, centerinfo, on="center_id", how="outer")
```

```
print("Shape of test data : ", test.shape)

test.head()

# Typecasting to assign appropriate data type to variables

col_names = ['center_id', 'meal_id', 'category', 'cuisine', 'city_code',
'region_code', 'center_type']

train[col_names] = train[col_names].astype('category')

test[col_names] = test[col_names].astype('category')

print("Train Datatype\n", train.dtypes)

print("Test Datatype\n", test.dtypes)

# Orders by centers

center_orders = train.groupby("center_id", as_index=False).sum()

center_orders = center_orders[["center_id",
"num_orders"]].sort_values(by="num_orders", ascending=False).head(10)

fig = px.bar(x=center_orders["center_id"].astype("str"),
y=center_orders["num_orders"], title="Top 10 Centers by Order",

             labels={"x": "center_id", "y": "num_orders"})

fig.show()

# Pie chart on food category

fig = px.pie(values=train["category"].value_counts(),
names=train["category"].unique(),

             title="Most popular food category")

fig.show()

# Orders by Cuisine types

cuisine_orders = train.groupby(["cuisine"], as_index=False).sum()

cuisine_orders = cuisine_orders[["cuisine",
"num_orders"]].sort_values(by="num_orders", ascending=False)

fig = px.bar(cuisine_orders, x="cuisine", y="num_orders", title="orders by
cuisine")

fig.show()

# Impact of check-out price on order
```

```
train_sample = train.sample(frac=0.2)

fig = px.scatter(train_sample, x="checkout_price", y="num_orders", title="number of order change with checkout price")

fig.show()

sns.boxplot(train["checkout_price"])

# Orders weekly trend

week_orders = train.groupby(["week"], as_index=False).sum()

week_orders = week_orders[["week", "num_orders"]]

fig = px.line(week_orders, x="week", y="num_orders", markers=True, title="Order weekly trend")

fig.show()

# Deriving discount percent and discount y/n

train['discount percent'] = ((train['base_price'] - train['checkout_price']) / train['base_price']) * 100

# Discount Y/N

train['discount y/n'] = [1 if x > 0 else 0 for x in (train['base_price'] - train['checkout_price'])]

# Creating same feature in test dataset

test['discount percent'] = ((test['base_price'] - test['checkout_price']) / test['base_price']) * 100

test['discount y/n'] = [1 if x > 0 else 0 for x in (test['base_price'] - test['checkout_price'])]

train.head(2)

# Check for correlation between numeric features

plt.figure(figsize=(13, 13))

sns.heatmap(train.corr(), linewidths=.1, cmap='Reds', annot=True)

plt.title('Correlation Matrix')

plt.show()
```

```
# Define One hot encoding function

def one_hot_encode(features_to_encode, dataset):

    encoder = OneHotEncoder(sparse=False)

    encoder.fit(dataset[features_to_encode])

    encoded_cols = pd.DataFrame(encoder.transform(dataset[features_to_encode]),
columns=encoder.get_feature_names())

    dataset = dataset.drop(columns=features_to_encode)

    for cols in encoded_cols.columns:

        dataset[cols] = encoded_cols[cols]

    return dataset



# get list of categorical variables in data set

ls = train.select_dtypes(include='category').columns.values.tolist()

# Run one-hot encoding on all categorical variables

features_to_encode = ls

data = one_hot_encode(features_to_encode, train)

data = data.reset_index(drop=True)

# Train-Validation Data Split

y = data[["num_orders"]]

X = data.drop(["num_orders", "id", "base_price", "discount y/n"], axis=1)

X = X.replace((np.inf, -np.inf, np.nan), 0)  # replace nan and infinity values with
0

# 20% of train data is used for validation

X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.20,
random_state=100)

# Prepare test data post applying onehot encoding

OH_test = one_hot_encode(features_to_encode, test)

test_final = OH_test.drop(["id", "base_price", "discount y/n"], axis=1)
```

```
# Create pipeline for scaling and modeling

RF_pipe = make_pipeline(StandardScaler(), RandomForestRegressor(n_estimators=100,
max_depth=7))

# Build Model

RF_pipe.fit(X_train, y_train)

# Predict Value

RF_train_y_pred = RF_pipe.predict(X_val)

# Model Evaluation-

print('R Square:', RF_pipe.score(X_val, y_val))

print('RMSLE:', 100 * np.sqrt(metrics.mean_squared_log_error(y_val,
RF_train_y_pred)))

# Applying algorithm to predict orders

test_y_pred = RF_pipe.predict(test_final)

Result = pd.DataFrame(test_y_pred)

print(Result.values)

Result = pd.DataFrame(test_y_pred)

Submission = pd.DataFrame(columns=['id', 'num_orders'])

Submission['id'] = test['id']

Submission['num_orders'] = Result.values

Submission.to_csv('My submission.csv', index=False)

print(Submission.shape)

print(Submission.head())
```