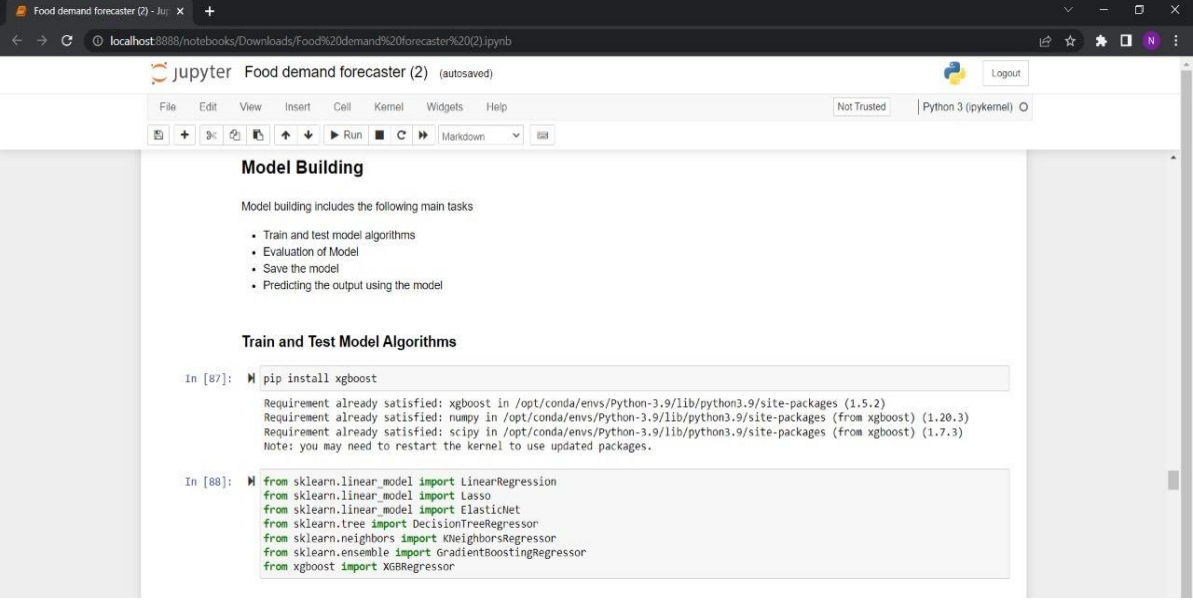


MODEL BUILDING

Team id: PNT2022TMID15311

TRAIN AND TEST MODEL ALGORITHMS



The screenshot shows a Jupyter Notebook titled "Food demand forecaster (2)". The "Model Building" section is active, containing a list of tasks and two code cells. The first code cell installs xgboost, and the second cell imports various machine learning models from sklearn.

```
Model Building
```

Model building includes the following main tasks

- Train and test model algorithms
- Evaluation of Model
- Save the model
- Predicting the output using the model

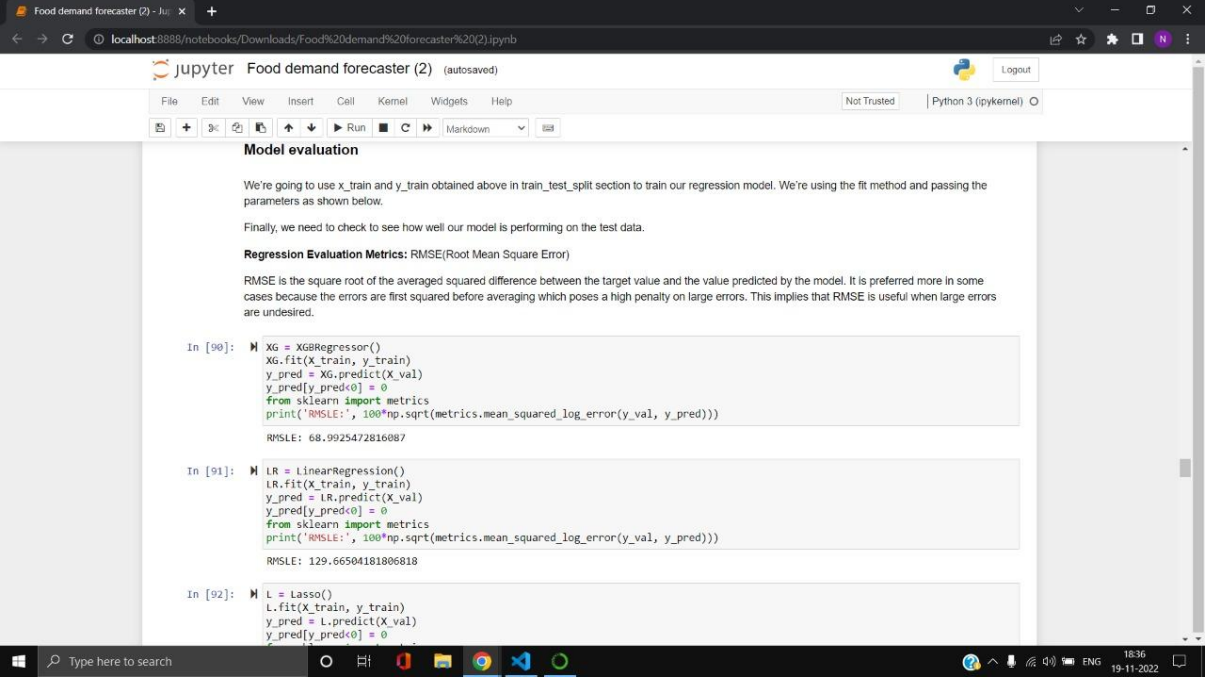
```
Train and Test Model Algorithms
```

```
In [87]: ! pip install xgboost
```

Requirement already satisfied: xgboost in /opt/conda/envs/Python-3.9/lib/python3.9/site-packages (1.5.2)
Requirement already satisfied: numpy in /opt/conda/envs/Python-3.9/lib/python3.9/site-packages (from xgboost) (1.20.3)
Requirement already satisfied: scipy in /opt/conda/envs/Python-3.9/lib/python3.9/site-packages (from xgboost) (1.7.3)
Note: you may need to restart the kernel to use updated packages.

```
In [88]: from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import GradientBoostingRegressor
from xgboost import XGBRegressor
```

MODEL EVALUATION



The screenshot shows the "Model Evaluation" section of the Jupyter Notebook. It explains the use of x_train and y_train for training and y_val for testing. It defines RMSE (Root Mean Square Error) and provides three code cells for evaluating XGBRegressor, LinearRegression, and Lasso models.

```
Model evaluation
```

We're going to use `x_train` and `y_train` obtained above in `train_test_split` section to train our regression model. We're using the `fit` method and passing the parameters as shown below.

Finally, we need to check to see how well our model is performing on the test data.

Regression Evaluation Metrics: RMSE(Root Mean Square Error)

RMSE is the square root of the averaged squared difference between the target value and the value predicted by the model. It is preferred more in some cases because the errors are first squared before averaging which poses a high penalty on large errors. This implies that RMSE is useful when large errors are undesired.

```
In [90]: XG = XGBRegressor()
XG.fit(X_train, y_train)
y_pred = XG.predict(X_val)
y_pred[y_pred<0] = 0
from sklearn import metrics
print('RMSE:', 100*np.sqrt(metrics.mean_squared_log_error(y_val, y_pred)))

RMSE: 68.9925472816087
```

```
In [91]: LR = LinearRegression()
LR.fit(X_train, y_train)
y_pred = LR.predict(X_val)
y_pred[y_pred<0] = 0
from sklearn import metrics
print('RMSE:', 100*np.sqrt(metrics.mean_squared_log_error(y_val, y_pred)))

RMSE: 129.66504181060818
```

```
In [92]: L = Lasso()
L.fit(X_train, y_train)
y_pred = L.predict(X_val)
y_pred[y_pred<0] = 0
```

```
Food demand forecaster (2) - Jupyter
localhost:8888/notebooks/Downloads/Food%20demand%20forecaster%20(2).ipynb

jupyter Food demand forecaster (2) (autosaved)
File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (pykernel)

print('RMSLE:', 100*np.sqrt(metrics.mean_squared_log_error(y_val, y_pred)))
RMSLE: 129.66504181806818

In [92]: L = Lasso()
L.fit(X_train, y_train)
y_pred = L.predict(X_val)
y_pred[y_pred<0] = 0
from sklearn import metrics
print('RMSLE:', 100*np.sqrt(metrics.mean_squared_log_error(y_val, y_pred)))
RMSLE: 129.22939048353018

In [93]: EN = ElasticNet()
EN.fit(X_train, y_train)
y_pred = EN.predict(X_val)
y_pred[y_pred<0] = 0
from sklearn import metrics
print('RMSLE:', 100*np.sqrt(metrics.mean_squared_log_error(y_val, y_pred)))
RMSLE: 131.12700245932322

In [94]: DT = DecisionTreeRegressor()
DT.fit(X_train, y_train)
y_pred = DT.predict(X_val)
y_pred[y_pred<0] = 0
from sklearn import metrics
print('RMSLE:', 100*np.sqrt(metrics.mean_squared_log_error(y_val, y_pred)))
RMSLE: 62.843557611229585

In [95]: KNN = KNeighborsRegressor()
KNN.fit(X_train, y_train)
y_pred = KNN.predict(X_val)
y_pred[y_pred<0] = 0
from sklearn import metrics
print('RMSLE:', 100*np.sqrt(metrics.mean_squared_log_error(y_val, y_pred)))
```

```
Food demand forecaster (2) - Jupyter
localhost:8888/notebooks/Downloads/Food%20demand%20forecaster%20(2).ipynb

jupyter Food demand forecaster (2) (autosaved)
File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (pykernel)

RMSLE: 62.843557611229585

In [95]: KNN = KNeighborsRegressor()
KNN.fit(X_train, y_train)
y_pred = KNN.predict(X_val)
y_pred[y_pred<0] = 0
from sklearn import metrics
print('RMSLE:', 100*np.sqrt(metrics.mean_squared_log_error(y_val, y_pred)))
RMSLE: 67.18681895455414

In [96]: GB = GradientBoostingRegressor()
GB.fit(X_train, y_train)
y_pred = GB.predict(X_val)
y_pred[y_pred<0] = 0
from sklearn import metrics
print('RMSLE:', 100*np.sqrt(metrics.mean_squared_log_error(y_val, y_pred)))
RMSLE: 97.77041683634049
```

SAVE THE MODEL

```
Save the Model

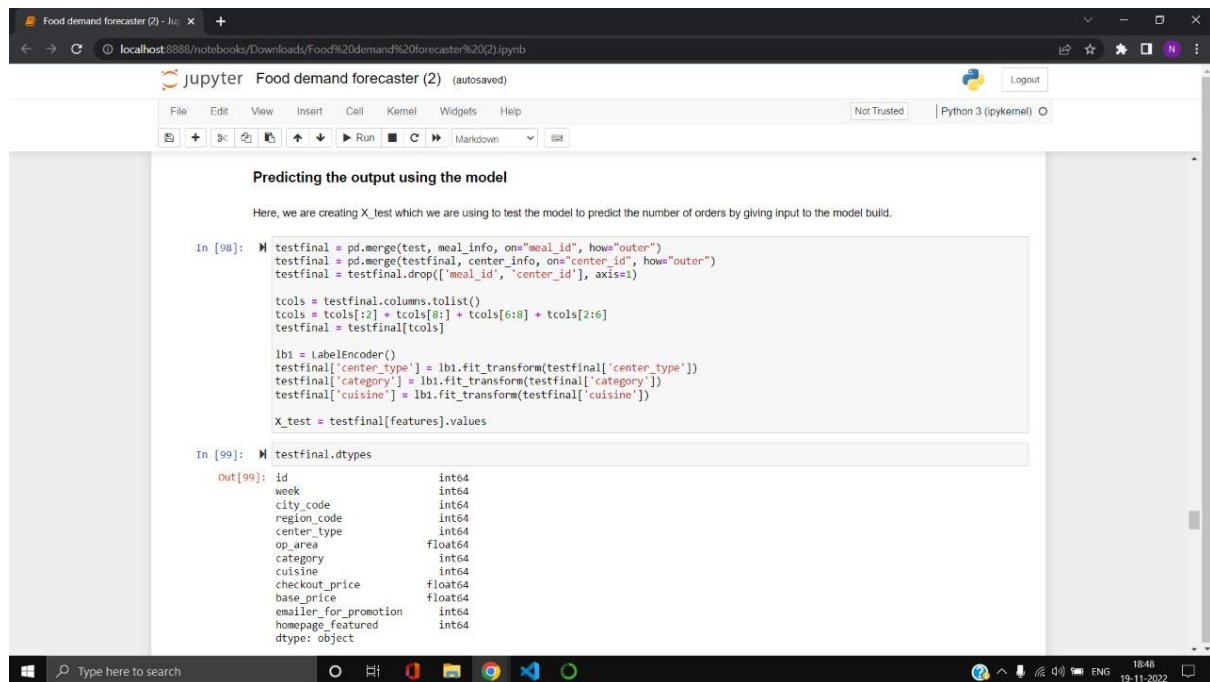
In [97]: import pickle
pickle.dump(DT, open('fdemand.pkl', 'wb'))

Predicting the output using the model

Here, we are creating X_test which we are using to test the model to predict the number of orders by giving input to the model build.

In [98]: testfinal = pd.merge(test, meal_info, on="meal_id", how="outer")
testfinal = pd.merge(testfinal, center_info, on="center_id", how="outer")
testfinal = testfinal.drop(['meal_id', 'center_id'], axis=1)
```

PREDICTING THE OUTPUT USING THE MODEL



The screenshot shows a Jupyter Notebook titled "Food demand forecaster (2)" with the following code and output:

```
In [98]: testfinal = pd.merge(test, meal_info, on="meal_id", how="outer")
testfinal = pd.merge(testfinal, center_info, on="center_id", how="outer")
testfinal = testfinal.drop(["meal_id", "center_id"], axis=1)

tcols = testfinal.columns.tolist()
tcols = tcols[2:] + tcols[8:] + tcols[6:8] + tcols[2:6]
testfinal = testfinal[tcols]

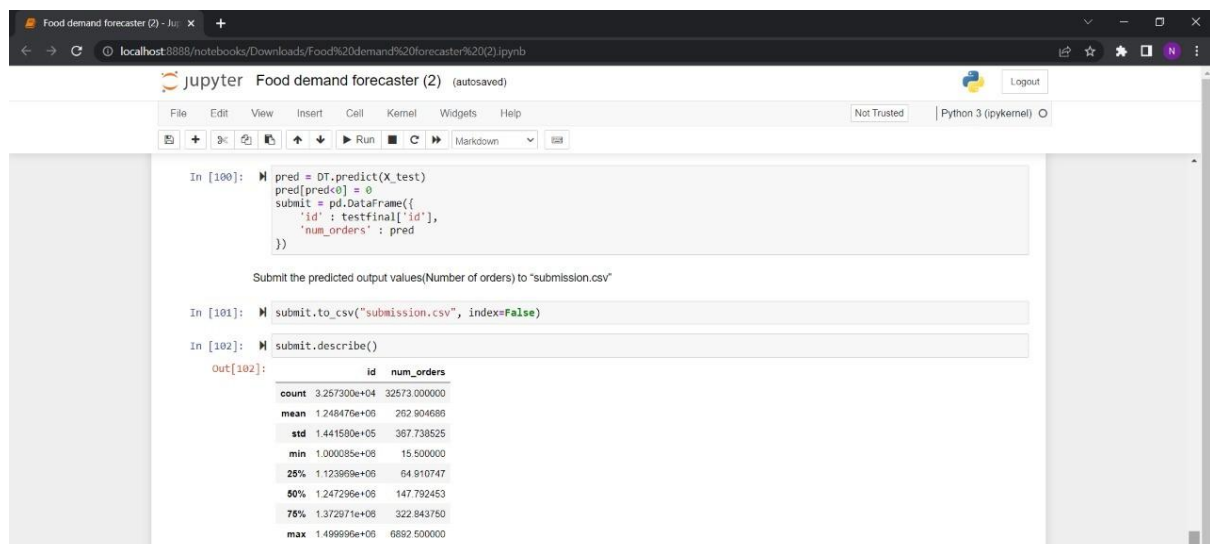
lbi = LabelEncoder()
testfinal['center_type'] = lbi.fit_transform(testfinal['center_type'])
testfinal['category'] = lbi.fit_transform(testfinal['category'])
testfinal['cuisine'] = lbi.fit_transform(testfinal['cuisine'])

X_test = testfinal[features].values

In [99]: testfinal.dtypes
```

Out[99]:

id		int64
week		int64
city_code		int64
region_code		int64
center_type		int64
op_area		float64
category		int64
cuisine		int64
checkout_price		float64
base_price		float64
emailer_for_promotion		int64
homepage_featured		int64
dtype:		object



The screenshot shows the continuation of the Jupyter Notebook with the following code and output:

```
In [100]: pred = DT.predict(X_test)
pred[pred<0] = 0
submit = pd.DataFrame({
    'id': testfinal['id'],
    'num_orders': pred
})

Submit the predicted output values(Number of orders) to "submission.csv"

In [101]: submit.to_csv("submission.csv", index=False)

In [102]: submit.describe()
```

Out[102]:

	id	num_orders
count	3.257300e+04	32573.000000
mean	1.248476e+06	262.904686
std	1.441580e+05	367.738525
min	1.000085e+06	15.500000
25%	1.123699e+06	64.910747
50%	1.247296e+06	147.792453
75%	1.372971e+06	322.843750
max	1.499996e+06	6892.500000