In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

In [3]:
```
df = pd.read_csv('rideshare_kaggle.csv')
df.head(5)
```

Out[3]:

	id	timestamp	hour	day	month	datetime	timezone	source	destination	cab_type	...	precipIntensit
0	424553bb-7174-41ea-aeb4-fe06d4f4b9d7	1.544953e+09	9	16	12	2018-12-16 09:30:07	America/New_York	Haymarket Square	North Station	Lyft	...	0
1	4bd23055-6827-41c6-b23b-3c491f24e74d	1.543284e+09	2	27	11	2018-11-27 02:00:23	America/New_York	Haymarket Square	North Station	Lyft	...	0
2	981a3613-77af-4620-a42a-0c0866077d1e	1.543367e+09	1	28	11	2018-11-28 01:00:22	America/New_York	Haymarket Square	North Station	Lyft	...	0
3	c2d88af2-d278-4bfd-a8d0-29ca77cc5512	1.543554e+09	4	30	11	2018-11-30 04:53:02	America/New_York	Haymarket Square	North Station	Lyft	...	0.
4	e0126e1f-8ca9-4f2e-82b3-50505a09db9a	1.543463e+09	3	29	11	2018-11-29 03:49:20	America/New_York	Haymarket Square	North Station	Lyft	...	0

5 rows × 57 columns

In [4]:
```
from sklearn.model_selection import train_test_split
# split the whole data into 66% training data and 33% test data
X = df.drop(columns=['price'])
y = df.price
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
```

In [5]:
```
train_data = X_train.merge(y_train, left_index = True, right_index = True)
# train_data.to_csv('Train_Data.csv') for export dataset use
```

In [6]:
```
train_data.shape
```

Out[6]:
```
(464357, 57)
```

In [7]:
```
test_data = X_test.merge(y_test, left_index = True, right_index = True)
# test_data.to_csv('Test_Data.csv') for export dataset use
```

In [8]:
```
df = train_data.dropna()
```

In [9]:
```
df.head(2)
```

Out[9]:

	id	timestamp	hour	day	month	datetime	timezone	source	destination	cab_type	...	uvIndexTim
611375	c752033c-1070-4ef5-99c5-7e0285824d68	1.543570e+09	9	30	11	2018-11-30 09:33:02	America/New_York	North Station	North End	Uber	...	154359360
562720	afbdd36a-0789-430c-81a8-50b13df927a8	1.544902e+09	19	15	12	2018-12-15 19:20:09	America/New_York	North Station	North End	Uber	...	154489320

2 rows × 57 columns

EDA

In [10]:
```
raw_df = pd.read_csv('rideshare_kaggle.csv')
raw_df['datetime'].min(),raw_df['datetime'].max()
```

Out[10]:
```
('2018-11-26 03:40:46', '2018-12-18 19:15:10')
```

In [11]:
```
# looking the number of unique value in each column
unique_dict = {}
for i in df.columns:
    unique_dict[i] = len(df[i].unique())
unique_dict
```

Out[11]:
```
{'id': 427416,
 'timestamp': 34314,
 'hour': 24,
 'day': 17,
 'month': 2,
 'datetime': 31333,
 'timezone': 1,
 'source': 12,
 'destination': 12,
 'cab_type': 2,
 'product_id': 12,
 'name': 12,
 'distance': 549,
 'surge_multiplier': 7,
 'latitude': 11,
 'longitude': 12,
 'temperature': 308,
 'apparentTemperature': 319,
 'short_summary': 9,
 'long_summary': 11,
 'precipIntensity': 63,
 'precipProbability': 29,
 'humidity': 51,
 'windSpeed': 291,
 'windGust': 286,
 'windGustTime': 25,
 'visibility': 227,
 'temperatureHigh': 129,
 'temperatureHighTime': 23,
 'temperatureLow': 133,
 'temperatureLowTime': 31,
 'apparentTemperatureHigh': 124,
 'apparentTemperatureHighTime': 27,
 'apparentTemperatureLow': 136,
 'apparentTemperatureLowTime': 32,
 'icon': 7,
 'dewPoint': 313,
 'pressure': 316,
 'windBearing': 195,
 'cloudCover': 83,
 'uvIndex': 3,
 'visibility.1': 227,
 'ozone': 274,
```

```
 'sunriseTime': 110,
 'sunsetTime': 114,
 'moonPhase': 18,
 'precipIntensityMax': 65,
 'uvIndexTime': 20,
 'temperatureMin': 131,
 'temperatureMinTime': 25,
 'temperatureMax': 128,
 'temperatureMaxTime': 23,
 'apparentTemperatureMin': 137,
 'apparentTemperatureMinTime': 29,
 'apparentTemperatureMax': 125,
 'apparentTemperatureMaxTime': 27,
 'price': 140}
```

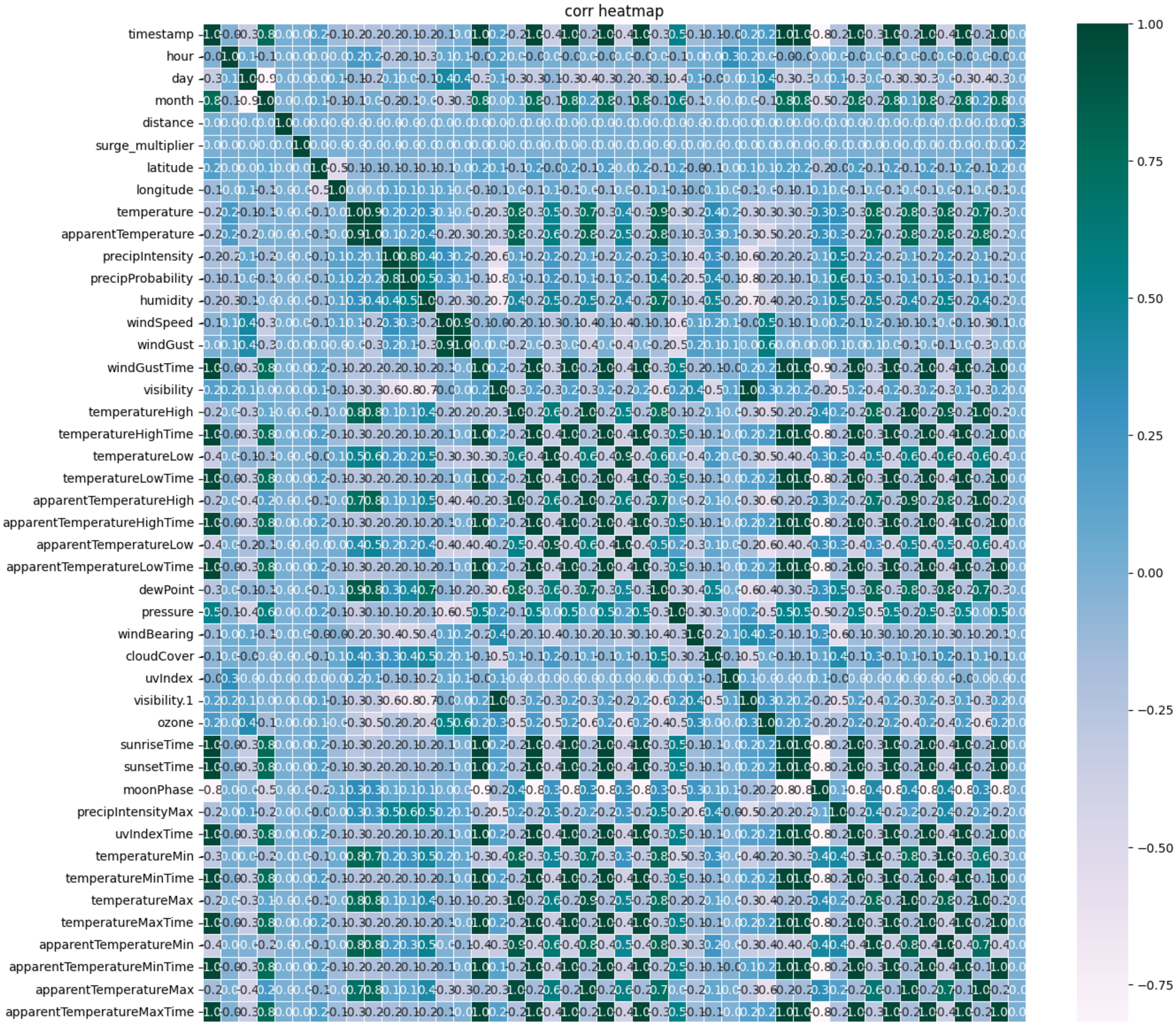
In [12]:
```
# There are 45 numerical features except the price
len(df._get_numeric_data().columns)
```

Out[12]:
```
46
```

In [13]:
```
categorical_cols=df.columns[df.dtypes =='object']
# There are 11 categorical features
categorical_cols
```

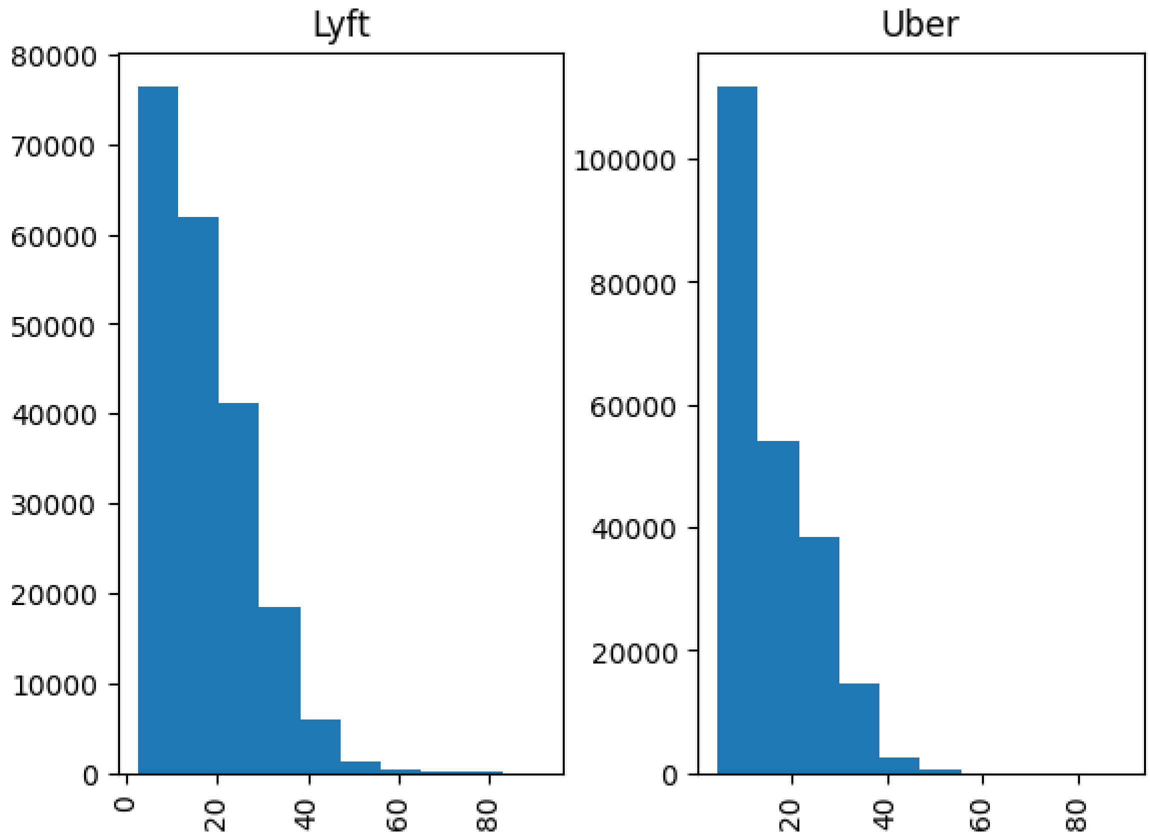
Out[13]:
```
Index(['id', 'datetime', 'timezone', 'source', 'destination', 'cab_type',
       'product_id', 'name', 'short_summary', 'long_summary', 'icon'],
      dtype='object')
```

In [14]:
```
corr_all = df.corr()
fig, ax = plt.subplots(figsize=(15,15))
# Create the heatmap
sns.heatmap(corr_all, cmap="PuBuGn", annot=True, fmt=".1f", linewidths=.5, ax=ax)
# Set the title and axis labels
ax.set_title("corr heatmap")
# Show the plot
plt.show()
```


corr heatmap
timestamp
hour
day
month
distance
surge_multiplier
latitude
longitude
temperature
apparentTemperature
precipIntensity
precipProbability
humidity
windSpeed
windGust
windGustTime
visibility
temperatureHigh
temperatureHighTime
temperatureLow
temperatureLowTime
apparentTemperatureHigh
apparentTemperatureHighTime
apparentTemperatureLow
apparentTemperatureLowTime
dewPoint
pressure
windBearing
cloudCover
uvIndex
visibility.1
ozone
sunriseTime
sunsetTime
moonPhase
precipIntensityMax
uvIndexTime
temperatureMin
temperatureMinTime
temperatureMax
temperatureMaxTime
apparentTemperatureMin
apparentTemperatureMinTime
apparentTemperatureMax
apparentTemperatureMaxTime
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75

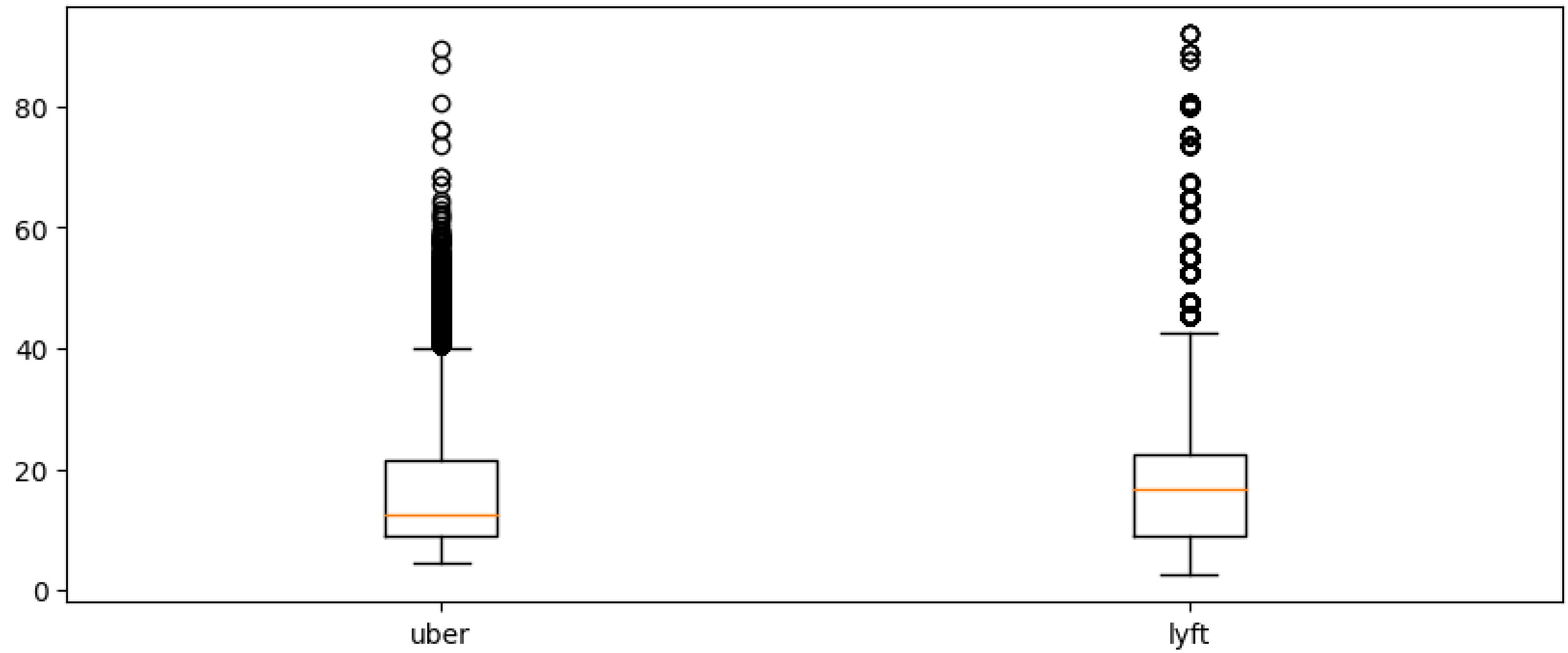
In [15]:
```
# Transform the column timestamp into panda datetime format
df['timestamp'] = pd.to_datetime(df['timestamp'],unit='s')
# create new column weekday
df['weekday'] = df['timestamp'].dt.strftime('%a')
df_feature = ['weekday','hour', 'cab_type', 'name', 'distance', 'surge_multiplier',
                     'long_summary', 'destination','price']
df = df[df_feature]
```

In [16]:
```
# Figure1
df[['cab_type','price']].hist(by="cab_type")
fig = plt.figure(figsize =(3,3))
plt.show()
```



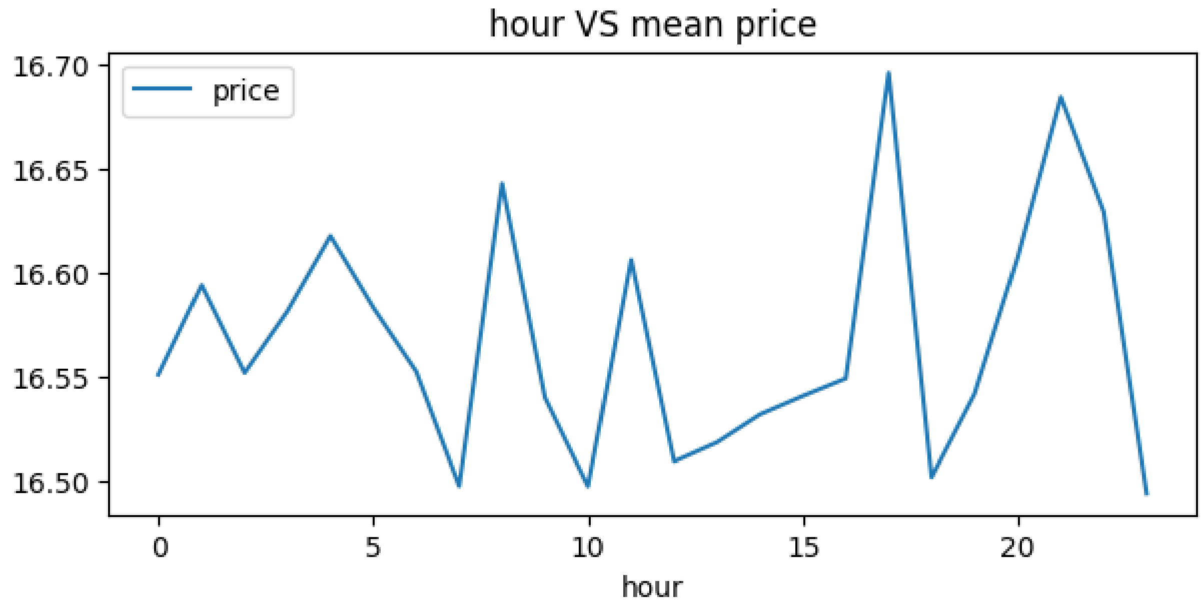

```
<Figure size 300x300 with 0 Axes>
```

In [17]:
```python
# Figure 2
uber = df.loc[df.cab_type == 'Uber']
lyft = df.loc[df.cab_type == 'Lyft']
fig = plt.figure(figsize =(10,4))
plt.boxplot([uber['price'], lyft['price']], labels=('uber', 'lyft'))
# show plot
plt.show()
```



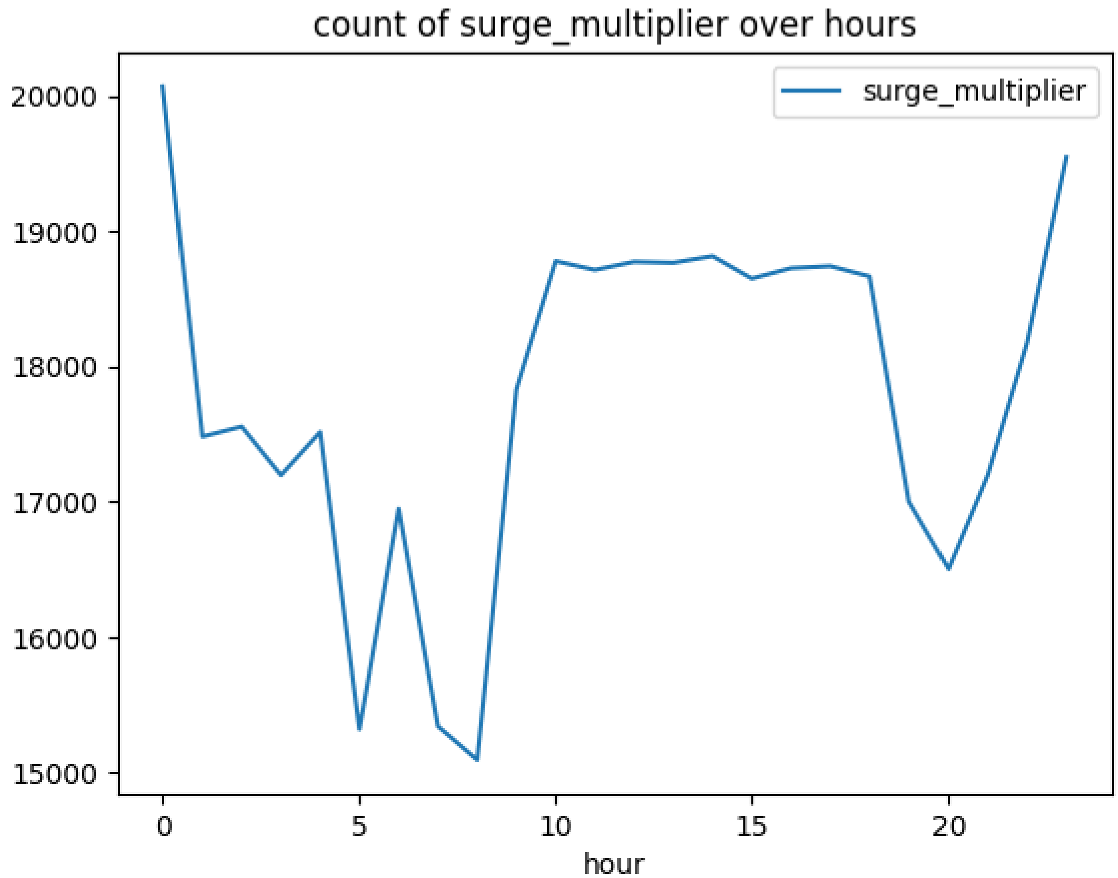
In [18]:
```
# Figure 3
df_hour_price = df.loc[:,['hour','price']].groupby(['hour']).mean()
df_hour_price.plot(kind ='line',figsize=(7, 3),title = 'hour VS mean price')
```

Out[18]:
```
<AxesSubplot:title={'center':'hour VS mean price'}, xlabel='hour'>
```



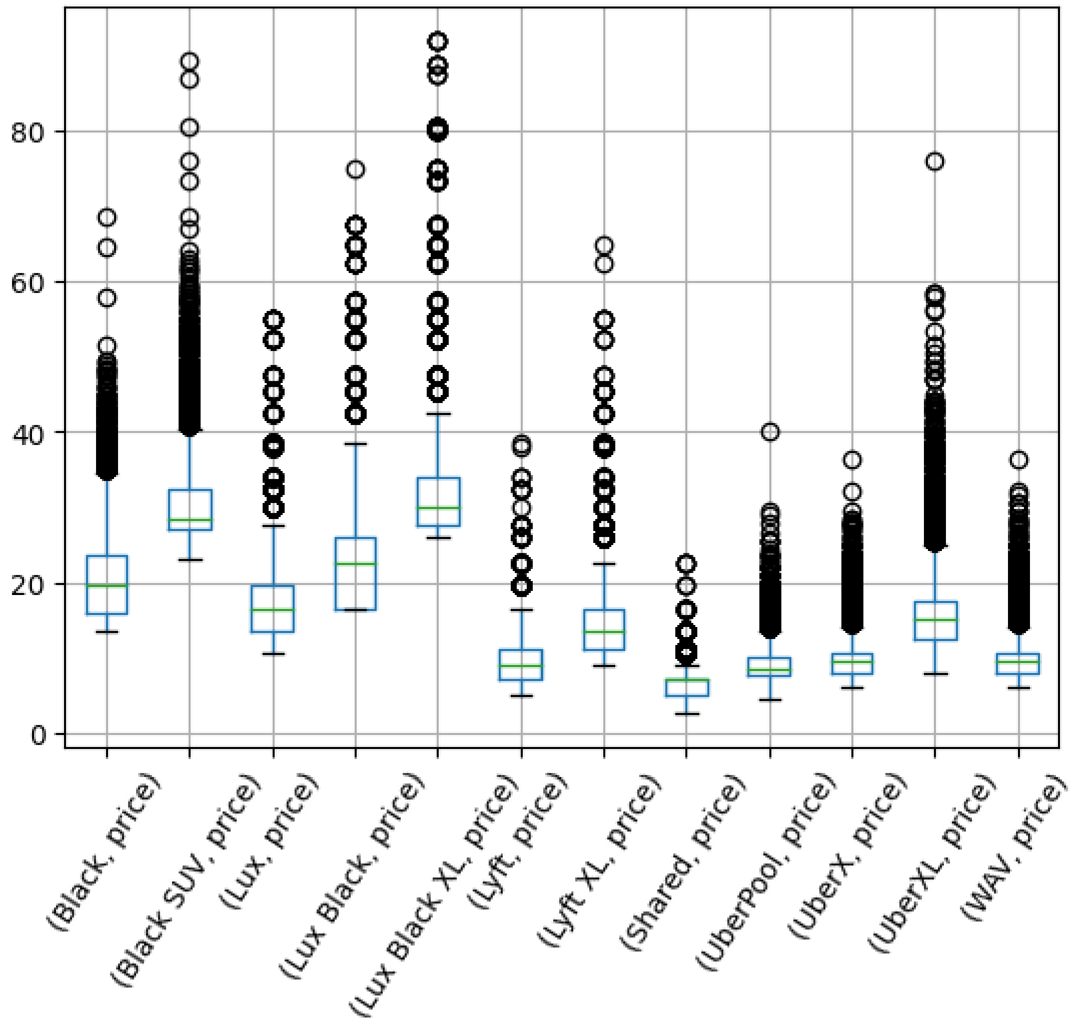
In [19]:
```
# Figure 4
df_surge_hour = df[['hour','surge_multiplier']]
hour_vs_multiplier = df_surge_hour.groupby(['hour']).count()
hour_vs_multiplier.plot(title = 'count of surge_multiplier over hours')
```

Out[19]: <AxesSubplot:title={'center':'count of surge_multiplier over hours'}, xlabel='hour'>



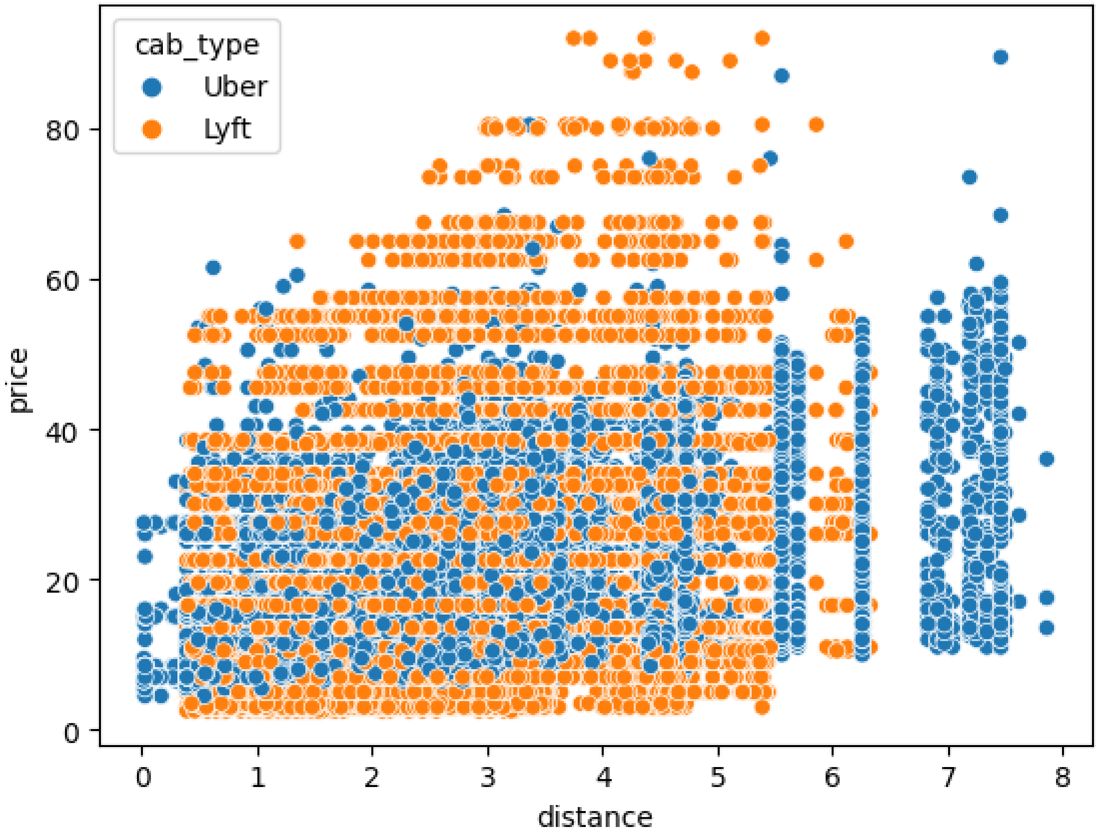
In [20]:
```
# Figure 5
name_price = df[['name','price']]
name_price.groupby(['name']).boxplot(subplots=False, rot=57, fontsize=10)
```

Out[20]: <AxesSubplot:>

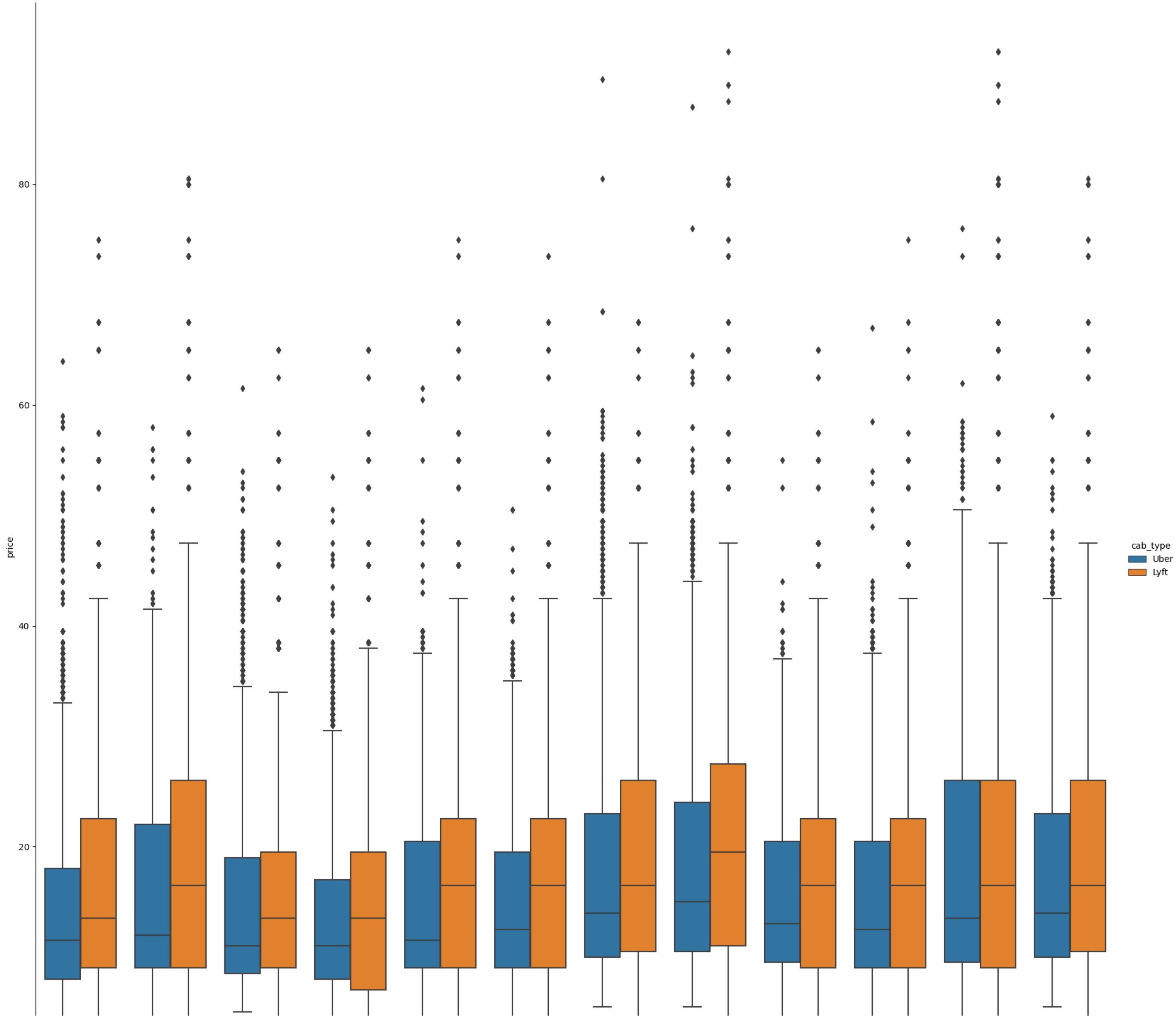


In [21]:
```
# Figure 6
sns.scatterplot(data=df, x="distance", y="price", hue='cab_type', sizes=(10,5))
```

Out[21]:
```
<AxesSubplot:xlabel='distance', ylabel='price'>
```



In [22]:
```
# Figure 7
sns.catplot(data=df, x="destination", y="price", hue="cab_type", kind="box", height=18)
fig = plt.figure(figsize =(6,3))
```


80
60
40
20
price
cab_type
Uber
Lyft

```
<Figure size 600x300 with 0 Axes>
```

Predictive Task & Baseline

In [24]:
```
data_raw = pd.read_csv('Train_Data.csv').drop(columns=['Unnamed: 0'])
data_raw.head(3)
```

Out[24]:

	id	timestamp	hour	day	month	datetime	timezone	source	destination	cab_type	...	uvIndexTime
0	c752033c-1070-4ef5-99c5-7e0285824d68	1.543570e+09	9	30	11	2018-11-30 09:33:02	America/New_York	North Station	North End	Uber	...	1543593600
1	afbdd36a-0789-430c-81a8-50b13df927a8	1.544902e+09	19	15	12	2018-12-15 19:20:09	America/New_York	North Station	North End	Uber	...	1544893200
2	1ae9c6a8-0df0-4ebd-a694-d559d5822005	1.543318e+09	11	27	11	2018-11-27 11:33:23	America/New_York	Haymarket Square	North Station	Lyft	...	1543338000

3 rows × 57 columns

In [25]:
```
# data preprocessing
data_raw['timestamp'] = pd.to_datetime(data_raw['timestamp'],unit='s')
data_raw['weekday'] = data_raw['timestamp'].dt.strftime('%a')
data_raw['price'].shape
```

Out[25]:
```
(464357,)
```

In [26]:
```
# select feature and drop null value
data = data_raw[['weekday','hour', 'cab_type', 'name', 'distance', 'surge_multiplier',
                 'long_summary', 'destination','price']].dropna()
# check size after drop null value
data.shape
```

Out[26]: (427416, 9)

In [27]:
```
data.columns
```

Out[27]:
```
Index(['weekday', 'hour', 'cab_type', 'name', 'distance', 'surge_multiplier',
       'long_summary', 'destination', 'price'],
      dtype='object')
```

In [28]:
```
X = data.drop(columns=['price'])
y = data['price']
```

In [29]:
```
# split the raw data into training data and test data
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3,random_state=42)
```

In [30]:
```
from sklearn.preprocessing import OrdinalEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
```

In [31]:
```
# data preprocessing
def df_transform(df):
    temp=df.copy()
    x= temp["long_summary"]
    condition=[x.str.lower().str.contains("cloudy"),x.str.lower().str.contains("rain"),x.str.lower().str.cont
    chioce=['cloudy','rain', 'foggy']
    temp["long_summary"]=np.select(condition, chioce, 'other')

    oen = OrdinalEncoder()
    temp[['weekday','hour', "cab_type",'name','long_summary','destination']]=oen.fit_transform(temp[['weekday
    return temp.drop(columns=['surge_multiplier'])
# data transformation
X_train=df_transform(X_train)
X_test=df_transform(X_test)
```

In [32]:
```
X_train.head(3)
```

Out[32]:

	weekday	hour	cab_type	name	distance	long_summary	destination
187161	1.0	1.0	1.0	9.0	2.79	3.0	1.0
462849	5.0	1.0	1.0	0.0	2.37	3.0	3.0
106050	6.0	13.0	0.0	5.0	3.39	0.0	8.0

In [33]:
```
X_test.head(3)
```

Out[33]:

	weekday	hour	cab_type	name	distance	long_summary	destination
96166	6.0	21.0	1.0	10.0	1.21	0.0	6.0
141614	1.0	1.0	1.0	8.0	1.82	0.0	0.0
68377	0.0	2.0	1.0	10.0	2.61	0.0	1.0

Baseline Implementation

In [34]:
```
from xgboost import XGBRegressor
xgb=XGBRegressor()
xgb.fit(X_train, y_train)
y_pred_xgb=xgb.predict(X_test)
rmse = np.sqrt(mean_squared_error(y_test,y_pred_xgb))
print('XGB RMSE:', rmse)

r2 = r2_score(y_test,y_pred_xgb)
print("R-squared:", r2)
```

```
XGB RMSE: 2.470667460488817
R-squared: 0.9299653080176011
```

In [35]:
```
from sklearn import linear_model
lr = linear_model.LinearRegression()
lr.fit(X_train, y_train)
y_pred_linear=lr.predict(X_test)
rmse = np.sqrt(mean_squared_error(y_test,y_pred_linear))
print('Linear RMSE:', rmse)

r2 = r2_score(y_test,y_pred_linear)
print("R-squared:", r2)
```

```
Linear RMSE: 6.795093990622744
R-squared: 0.4702443849219795
```

In [36]:

```
lasso = linear_model.Lasso(alpha=0.1)
lasso.fit(X_train, y_train)
lasso_y_pred=lasso.predict(X_test)
rmse = np.sqrt(mean_squared_error(y_test,lasso_y_pred))
print('Lasso RMSE:', rmse)

r2 = r2_score(y_test,lasso_y_pred)
print("R-squared:", r2)
```

```
Lasso RMSE: 6.800371402227667
R-squared: 0.46942119569453
```

In [37]:

```
ridge = linear_model.Ridge(alpha=0.1)
ridge.fit(X_train, y_train)
ridge_y_pred=ridge.predict(X_test)
rmse = np.sqrt(mean_squared_error(y_test,ridge_y_pred))
print('Ridege RMSE:', rmse)

r2 = r2_score(y_test,ridge_y_pred)
print("R-squared:", r2)
```

```
Ridege RMSE: 6.795093993767946
R-squared: 0.47024438443157046
```

In [38]:

```
from sklearn.ensemble import RandomForestRegressor
rfr = RandomForestRegressor()
rfr.fit(X_train, y_train)
y_pred_rfr = rfr.predict(X_test)

rmse = np.sqrt(mean_squared_error(y_test,y_pred_rfr))
print('RandomForest RMSE:', rmse)

r2 = r2_score(y_test,y_pred_rfr)
print("R-squared:", r2)
```

```
RandomForest RMSE: 2.759018147132373
R-squared: 0.9126639097615398
```

In [39]:

```
from sklearn.ensemble import GradientBoostingRegressor
gbr = GradientBoostingRegressor(random_state=0)
gbr.fit(X_train, y_train)
```

```
y_pred_gbr = gbr.predict(X_test)
rmse = np.sqrt(mean_squared_error(y_test,y_pred_gbr))
print('GradientBoosting RMSE:', rmse)

r2 = r2_score(y_test,y_pred_gbr)
print("R-squared:", r2)
```

```
GradientBoosting RMSE: 2.636234099406946
R-squared: 0.920264342720314
```

In [40]:
```
sdg = linear_model.SGDRegressor(alpha=0.1)
sdg.fit(X_train, y_train)
sdg_y_pred=sdg.predict(X_test)
rmse = np.sqrt(mean_squared_error(y_test,sdg_y_pred))
print('SGDRegressor RMSE:', rmse)

r2 = r2_score(y_test,sdg_y_pred)
print("R-squared:", r2)
```

```
SGDRegressor RMSE: 6.988459055507356
R-squared: 0.4396653463206569
```

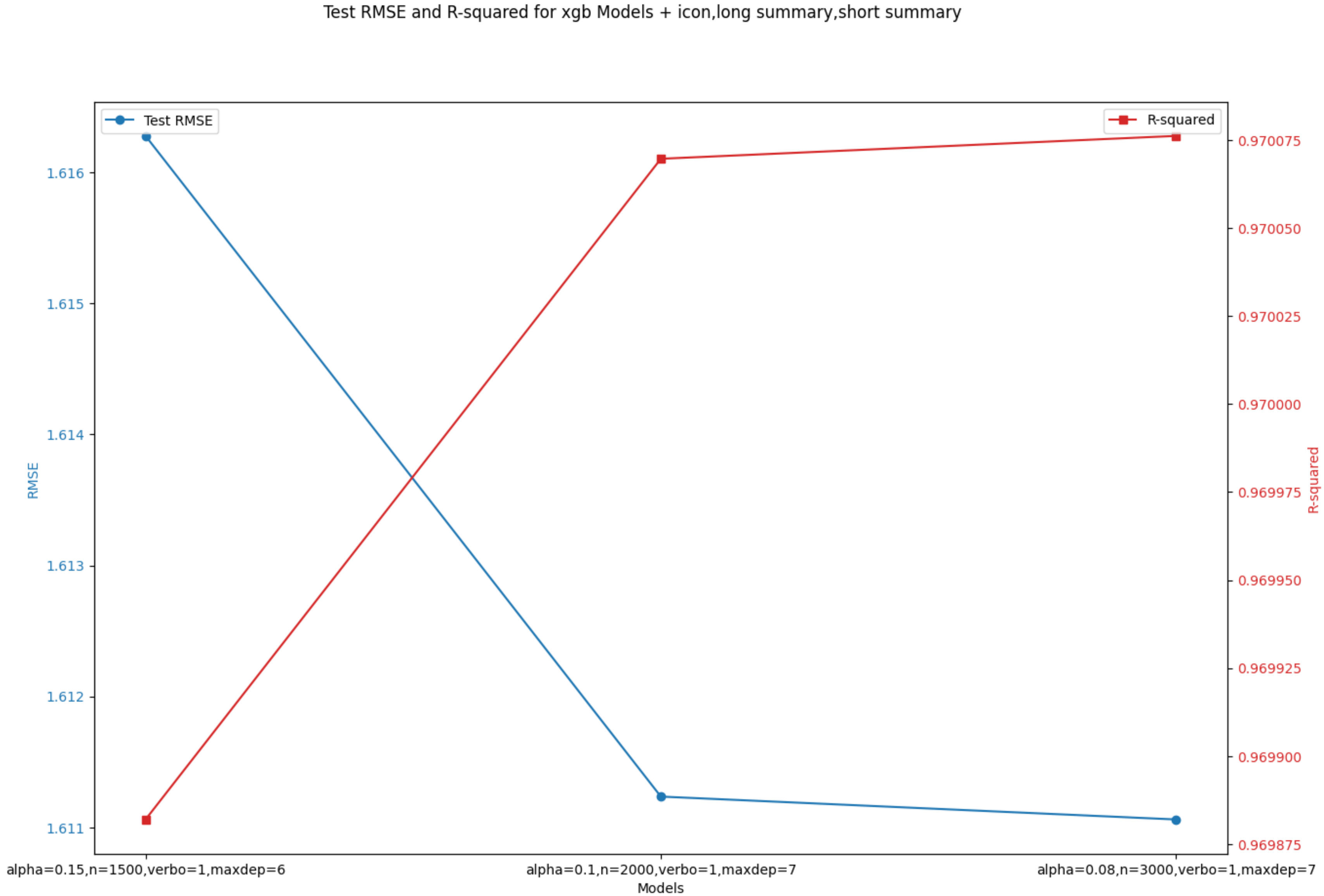
In [41]:
```
from sklearn.ensemble import AdaBoostRegressor
abr = AdaBoostRegressor(random_state=0)
abr.fit(X_train, y_train)
abr_y_pred=abr.predict(X_test)
rmse = np.sqrt(mean_squared_error(y_test,abr_y_pred))
print('AdaBoostRegressor RMSE:', rmse)

r2 = r2_score(y_test,abr_y_pred)
print("R-squared:", r2)
```

```
AdaBoostRegressor RMSE: 5.405862455450146
R-squared: 0.6647146212342192
```

Visualization of Performance

In [22]:
```
import matplotlib.pyplot as plt

models = ['alpha=0.15,n=1500,verbo=1,maxdep=6',
          'alpha=0.1,n=2000,verbo=1,maxdep=7',
          'alpha=0.08,n=3000,verbo=1,maxdep=7']
test_rmse = [1.6162792236901764, 1.6112352558011844, 1.6110603602622964]
```

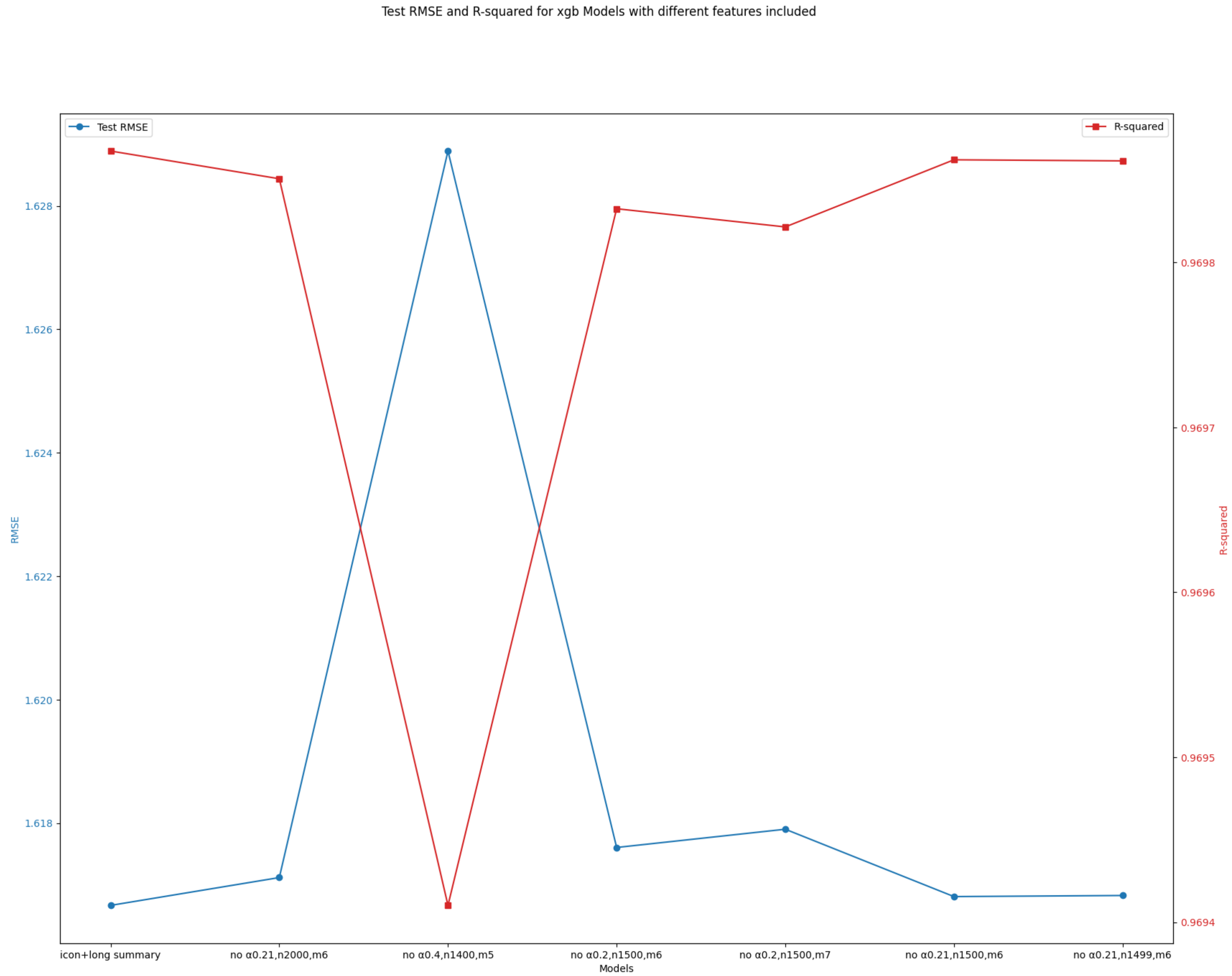
```
r_squared = [0.9698819853546095, 0.9700696723024124, 0.9700761696735035]

fig, ax1 = plt.subplots(figsize=(15, 10))

# 绘制测试 RMSE 折线
ax1.plot(models, test_rmse, marker='o', label='Test RMSE', color='tab:blue')
ax1.set_xlabel('Models')
ax1.set_ylabel('RMSE', color='tab:blue')
ax1.tick_params(axis='y', labelcolor='tab:blue')

# 创建共享 x 轴的第二个 y 轴
ax2 = ax1.twinx()

# 绘制 R-squared 折线
ax2.plot(models, r_squared, marker='s', label='R-squared', color='tab:red')
ax2.set_ylabel('R-squared', color='tab:red')
ax2.tick_params(axis='y', labelcolor='tab:red')

# 添加标题和图例
fig.suptitle('Test RMSE and R-squared for xgb Models + icon,long summary,short summary')
ax1.legend(loc='upper left')
ax2.legend(loc='upper right')

plt.show()
```



In [23]:

```
import matplotlib.pyplot as plt

models = ['icon+long summary',
          'no α0.21,n2000,m6',
          'no α0.4,n1400,m5',
          'no α0.2,n1500,m6',
          'no α0.2,n1500,m7',
```

```
          'no α0.21,n1500,m6',
          'no α0.21,n1499,m6'
         ]
test_rmse = [1.6166667451737495, 1.617117566422057, 1.6288902740018953,1.6176050519744807,1.6179007509430354,
r_squared = [0.9698675413442925, 0.9698507336172549, 0.9694101589918112,0.969832553681098,0.9698215234261124,

fig, ax1 = plt.subplots(figsize=(20, 15))

# 绘制测试 RMSE 折线
ax1.plot(models, test_rmse, marker='o', label='Test RMSE', color='tab:blue')
ax1.set_xlabel('Models')
ax1.set_ylabel('RMSE', color='tab:blue')
ax1.tick_params(axis='y', labelcolor='tab:blue')

# 创建共享 x 轴的第二个 y 轴
ax2 = ax1.twinx()

# 绘制 R-squared 折线
ax2.plot(models, r_squared, marker='s', label='R-squared', color='tab:red')
ax2.set_ylabel('R-squared', color='tab:red')
ax2.tick_params(axis='y', labelcolor='tab:red')

# 添加标题和图例
fig.suptitle('Test RMSE and R-squared for xgb Models with different features included')
ax1.legend(loc='upper left')
ax2.legend(loc='upper right')

plt.show()
```


Test RMSE and R-squared for xgb Models with different features included
Test RMSE
R-squared
RMSE
1.628
1.626
1.624
1.622
1.620
1.618
R-squared
0.9698
0.9697
0.9696
0.9695
0.9694
icon+long summary
no α0.21,n2000,m6
no α0.4,n1400,m5
no α0.2,n1500,m6
no α0.2,n1500,m7
no α0.21,n1500,m6
no α0.21,n1499,m6
Models

Final Model

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.pipeline import Pipeline
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.linear_model import ElasticNet
from sklearn.svm import SVR
from xgboost import XGBRegressor
from sklearn.preprocessing import OrdinalEncoder
from sklearn.model_selection import GridSearchCV
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [2]:
```
data = pd.read_csv('Train_Data.csv')
pd.set_option('display.max_columns', None)

data = data.dropna()

data['timestamp'] = pd.to_datetime(data['timestamp'],unit='s')
data['weekday'] = data['timestamp'].dt.strftime('%a')

test = pd.read_csv('Test_Data.csv')
test['timestamp'] = pd.to_datetime(test['timestamp'],unit='s')
test['weekday'] = test['timestamp'].dt.strftime('%a')
test = test[['weekday','hour', 'cab_type', 'name', 'distance',
             'surge_multiplier', 'icon','long_summary','short_summary', 'destination','price']].dropna()
test_y = test['price']
test_X = test.drop('price', axis=1)
test_X.reset_index(inplace=True)
```

```
test_X = test_X.drop('index', axis=1)
test_X
test_y = test_y.reset_index().drop('index', axis=1)
test_y
```

Out[2]:

	price
0	8.0
1	13.5
2	13.5
3	27.5
4	31.5
...	...
210555	26.0
210556	22.5
210557	7.0
210558	16.5
210559	10.5

210560 rows × 1 columns

In [4]:
```
data_feature = data[['weekday','hour', 'cab_type', 'name', 'distance', 'surge_multiplier', 'icon','long_summa
```

In [5]:
```
train_y = data_feature['price']
train_X = data_feature.drop('price',axis=1)
train_X.reset_index(inplace=True)
train_X = train_X.drop('index',axis=1)
train_y = train_y.reset_index().drop('index', axis=1)
train_y
```

Out[5]:

	price
0	15.0
1	16.0
2	11.0
3	8.5
4	16.5
...	...
427411	11.5
427412	16.5
427413	16.0
427414	18.5
427415	13.5

427416 rows × 1 columns

In [6]:
```
from sklearn.decomposition import TruncatedSVD,LatentDirichletAllocation
from sklearn.feature_extraction.text import TfidfVectorizer,TfidfTransformer,CountVectorizer
from sklearn.pipeline import Pipeline,make_pipeline
text_features = ['long_summary']
text_ff = Pipeline([('tfidf',TfidfVectorizer()), ('svd',TruncatedSVD(n_components=11))])
TfidfVectorizer().fit_transform(test_X[text_features[0]]).shape
text_ff.fit(test_X[text_features[0]])
TfidfVectorizer().fit_transform(test_X[text_features[0]]).shape
```

Out[6]: (210560, 21)

In [7]:
```
text_features = ['long_summary']
text_ff = Pipeline([('tfidf',TfidfVectorizer()), ('svd',TruncatedSVD(n_components=11))])
TfidfVectorizer().fit_transform(train_X[text_features[0]]).shape
text_ff.fit(train_X[text_features[0]])
TfidfVectorizer().fit_transform(train_X[text_features[0]]).shape
```

Out[7]: (427416, 21)

In [8]:
```
def text_feature():
    for i,col in enumerate(text_features):
        if i ==0:
            text_data = text_ff.fit_transform((test_X[col].astype(str)))
        else:
            text_data = np.concatenate((text,text_ff.fit_transform(test_X[col])))
    return text_data
```

In [9]:
```
def text_feature1():
    for i,col in enumerate(text_features):
        if i ==0:
            text_data = text_ff.fit_transform((train_X[col].astype(str)))
        else:
            text_data = np.concatenate((text,text_ff.fit_transform(train_X[col])))
    return text_data
```

In [10]:
```
text_train = text_feature1()
text_train.shape
```

Out[10]:
```
(427416, 11)
```

In [11]:
```
text_test = text_feature()
text_test
```

Out[11]:
```
array([[ 2.26488815e-01,  9.19097434e-01, -8.31610118e-03, ...,
         7.36847304e-03, -3.80491514e-03, -1.50911801e-04],
       [ 2.26488815e-01,  9.19097434e-01, -8.31610118e-03, ...,
         7.36847304e-03, -3.80491514e-03, -1.50911801e-04],
       [ 9.19594212e-01, -1.55680074e-01, -3.54706612e-01, ...,
        -2.02822559e-03, -2.12571855e-04,  5.46878429e-05],
       ...,
       [ 1.60802581e-01,  7.23827094e-01, -9.61215944e-03, ...,
        -1.21862216e-03,  8.77514470e-05, -2.89252249e-03],
       [ 9.19594212e-01, -1.55680074e-01, -3.54706612e-01, ...,
        -2.02822559e-03, -2.12571855e-04,  5.46878429e-05],
       [ 8.30679174e-01, -1.30550672e-01,  5.34606349e-01, ...,
        -2.29920910e-03, -2.37893056e-04,  6.10037482e-05]])
```

In [12]:
```
text_test = pd.DataFrame(text_test,columns=[f'text{i}' for i in range(11)])
text_test
```

Out[12]:

	text0	text1	text2	text3	text4	text5	text6	text7	text8	text9	text10
0	0.226489	0.919097	-0.008316	-0.148981	-0.091201	0.046408	-0.250468	-0.091780	0.007368	-0.003805	-0.000151
1	0.226489	0.919097	-0.008316	-0.148981	-0.091201	0.046408	-0.250468	-0.091780	0.007368	-0.003805	-0.000151
2	0.919594	-0.155680	-0.354707	-0.009773	0.031242	-0.051224	-0.024328	-0.001387	-0.002028	-0.000213	0.000055
3	0.919594	-0.155680	-0.354707	-0.009773	0.031242	-0.051224	-0.024328	-0.001387	-0.002028	-0.000213	0.000055
4	0.160803	0.723827	-0.009612	-0.236345	-0.257309	-0.423193	0.382083	0.054483	-0.001219	0.000088	-0.002893
...	...	...	...	...	...	...	...	...	...	...	...
210555	0.226489	0.919097	-0.008316	-0.148981	-0.091201	0.046408	-0.250468	-0.091780	0.007368	-0.003805	-0.000151
210556	0.830679	-0.130551	0.534606	-0.017423	0.043229	-0.063978	-0.029176	-0.001615	-0.002299	-0.000238	0.000061
210557	0.160803	0.723827	-0.009612	-0.236345	-0.257309	-0.423193	0.382083	0.054483	-0.001219	0.000088	-0.002893
210558	0.919594	-0.155680	-0.354707	-0.009773	0.031242	-0.051224	-0.024328	-0.001387	-0.002028	-0.000213	0.000055
210559	0.830679	-0.130551	0.534606	-0.017423	0.043229	-0.063978	-0.029176	-0.001615	-0.002299	-0.000238	0.000061

210560 rows × 11 columns

In [13]:

```
text_train = pd.DataFrame(text_train,columns=[f'text{i}' for i in range(11)])
text_train
```

Out[13]:

	text0	text1	text2	text3	text4	text5	text6	text7	text8	text9	text10
0	0.920119	-0.155749	-0.353309	-0.010245	0.031144	-0.051382	-0.023963	-0.001404	-0.002048	-0.000217	0.000062
1	0.920119	-0.155749	-0.353309	-0.010245	0.031144	-0.051382	-0.023963	-0.001404	-0.002048	-0.000217	0.000062
2	0.226438	0.919498	-0.008505	-0.145711	-0.093577	0.043388	-0.250722	-0.091498	0.007548	-0.003897	-0.000168
3	0.920119	-0.155749	-0.353309	-0.010245	0.031144	-0.051382	-0.023963	-0.001404	-0.002048	-0.000217	0.000062
4	0.920119	-0.155749	-0.353309	-0.010245	0.031144	-0.051382	-0.023963	-0.001404	-0.002048	-0.000217	0.000062
...	...	...	...	...	...	...	...	...	...	...	...
427411	0.829764	-0.130201	0.536034	-0.018405	0.043334	-0.064429	-0.028823	-0.001639	-0.002327	-0.000244	0.000069
427412	0.092553	0.346888	-0.002855	-0.104888	-0.048306	-0.076637	0.192564	-0.053442	-0.019126	0.018257	0.900694
427413	0.920119	-0.155749	-0.353309	-0.010245	0.031144	-0.051382	-0.023963	-0.001404	-0.002048	-0.000217	0.000062
427414	0.668616	0.257795	0.044671	0.171140	-0.359656	0.513412	0.243868	0.029268	-0.043951	0.002268	-0.000813
427415	0.920119	-0.155749	-0.353309	-0.010245	0.031144	-0.051382	-0.023963	-0.001404	-0.002048	-0.000217	0.000062

427416 rows × 11 columns

In [14]:

```
test_X = test_X.merge(text_test, left_index=True, right_index=True)


test_X.drop('long_summary',axis = 1,inplace = True)
test_X
```

Out[14]:

	weekday	hour	cab_type	name	distance	surge_multiplier	icon	short_summary	destination	text0	text1
0	Mon	22	Uber	UberX	1.89	1.0	cloudy	Overcast	Theatre District	0.226489	0.919097
1	Wed	0	Lyft	Lyft XL	1.97	1.0	clear-night	Clear	Theatre District	0.226489	0.919097
2	Sat	5	Lyft	Lux	1.23	1.0	cloudy	Overcast	West End	0.919594	-0.155680
3	Fri	10	Lyft	Lux	4.28	1.0	clear-night	Clear	Financial District	0.919594	-0.155680
4	Mon	17	Uber	Black SUV	2.34	1.0	cloudy	Overcast	Back Bay	0.160803	0.723827
...	...	...	...	...	...	...	...	...	...	...	...
210555	Mon	15	Lyft	Lux Black XL	1.49	1.0	cloudy	Overcast	Back Bay	0.226489	0.919097
210556	Thu	7	Lyft	Lux	2.99	1.0	partly-cloudy-night	Mostly Cloudy	Fenway	0.830679	-0.130551
210557	Sat	12	Uber	UberPool	1.21	1.0	partly-cloudy-day	Partly Cloudy	North End	0.160803	0.723827
210558	Tue	17	Lyft	Lyft XL	2.96	1.0	clear-day	Clear	Fenway	0.919594	-0.155680
210559	Thu	8	Lyft	Lyft	3.05	1.0	clear-night	Clear	Northeastern University	0.830679	-0.130551

210560 rows × 20 columns

In [15]:

```
train_X = train_X.merge(text_train, left_index=True, right_index=True)


train_X.drop('long_summary',axis = 1,inplace = True)
train_X
```

Out[15]:

	weekday	hour	cab_type	name	distance	surge_multiplier	icon	short_summary	destination	text0	text1	
0	Fri	9	Uber	Black	1.08	1.0	clear-night	Clear	North End	0.920119	-0.155749	-0
1	Sat	19	Uber	Black	1.19	1.0	clear-day	Clear	North End	0.920119	-0.155749	-0
2	Tue	11	Lyft	Lux	0.44	1.0	rain	Light Rain	North Station	0.226438	0.919498	-0
3	Tue	17	Uber	WAV	1.68	1.0	clear-day	Clear	South Station	0.920119	-0.155749	-0
4	Tue	9	Lyft	Lux Black	0.76	1.0	cloudy	Overcast	Haymarket Square	0.920119	-0.155749	-0
...	...	...	...	...	...	...	...	...	...	...	...	
427411	Fri	15	Uber	UberX	4.40	1.0	clear-day	Clear	Financial District	0.829764	-0.130201	0
427412	Mon	4	Lyft	Lux Black	0.91	1.0	cloudy	Overcast	Financial District	0.092553	0.346888	-0
427413	Fri	22	Uber	Black	0.65	1.0	cloudy	Overcast	Financial District	0.920119	-0.155749	-0
427414	Sun	10	Uber	UberXL	3.08	1.0	cloudy	Overcast	Northeastern University	0.668616	0.257795	(
427415	Sat	9	Lyft	Lyft XL	1.40	1.0	cloudy	Overcast	Fenway	0.920119	-0.155749	-0

427416 rows × 20 columns

In [17]:

```
test_X['name'].unique()
ordinal_enc = {'Shared': 1, 'UberPool': 1, 'Lyft':2, 'UberX': 3, 'WAV':3,
              "UberXL": 4, 'Lyft XL':4, 'Lux':5, 'Lux Black': 6, 'Black': 6,
              'Lux Black XL':7, 'Black SUV': 7}
test_X['name'] = test_X['name'].replace(ordinal_enc)
```

In [18]:

```
train_X['name'].unique()
ordinal_enc = {'Shared': 1, 'UberPool': 1, 'Lyft':2, 'UberX': 3, 'WAV':3,
              "UberXL": 4, 'Lyft XL':4, 'Lux':5, 'Lux Black': 6, 'Black': 6,
              'Lux Black XL':7, 'Black SUV': 7}
train_X['name'] = train_X['name'].replace(ordinal_enc)
```

In [ ]:
```
#chatgpt
preprocessor = ColumnTransformer(
    transformers=[
        ('onehot', OneHotEncoder(), ['weekday', 'cab_type', 'destination', 'icon', 'short_summary']),
        ('tfidf', TfidfVectorizer(), 'long_summary')
    ],
    remainder='passthrough'
)

xgb_regressor = XGBRegressor()

pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('regressor', xgb_regressor)
])

param_grid = {
    'regressor__learning_rate': [0.01, 0.2],
    'regressor__n_estimators': [500, 3000],
    'regressor__max_depth': [4, 7],
    'regressor__subsample': [0.5, 1],
    'regressor__colsample_bytree': [0.5, 1]
}

grid_search = GridSearchCV(pipeline, param_grid, scoring='neg_mean_squared_error', cv=5, n_jobs=-1, verbose=1
grid_search.fit(train_X, train_y)

print("Best parameters found: ", grid_search.best_params_)
print("Best score found: ", grid_search.best_score_)
```

```
Fitting 5 folds for each of 32 candidates, totalling 160 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
```

In [53]:
```
X = data_feature[['weekday','hour', 'cab_type', 'name', 'distance', 'surge_multiplier','destination']]
y = data_feature['price']
```

In [54]:
```
X = data_feature.drop('price',axis = 1)
y = data_feature['price']
```

In [19]:
```
numeric_transformer = 'passthrough' # passthrough for numeric features
categorical_transformer = Pipeline(steps=[
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
```

```
])

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, ['distance', 'surge_multiplier', 'text0', 'text1', 'text2', 'text3', 'te
       'text5', 'text6', 'text7', 'text8', 'text9', 'text10','name']),
        ('cat', categorical_transformer, ['weekday','cab_type','destination','icon','short_summary','hour'])
    ])
```

In [66]:
```
#最高#icon+long+short
xgb = Pipeline([('prepocessor',preprocessor)
                ,('xgb',XGBRegressor(learning_rate=0.15,
                                     n_estimators=1500, verbosity=1, max_depth=6))])
xgb.fit(train_X, train_y)
y_pred = xgb.predict(test_X)

rmse = np.sqrt(mean_squared_error(test_y,y_pred))
print('xgb RMSE:', rmse)
r2 = r2_score(test_y,y_pred)
print("R-squared:", r2)
```

```
xgb RMSE: 1.6162792236901764
R-squared: 0.9698819853546095
```

In [71]:
```
#最高#icon+long+short
xgb = Pipeline([('prepocessor',preprocessor)
                ,('xgb',XGBRegressor(learning_rate=0.1,
                                     n_estimators=2000, verbosity=1, max_depth=7))])
xgb.fit(train_X, train_y)
y_pred = xgb.predict(test_X)

rmse = np.sqrt(mean_squared_error(test_y,y_pred))
print('xgb RMSE:', rmse)
r2 = r2_score(test_y,y_pred)
print("R-squared:", r2)
```

```
xgb RMSE: 1.6112352558011844
R-squared: 0.9700696723024124
```

In [75]:
```
#最高#icon+long+short
xgb = Pipeline([('prepocessor',preprocessor)
                ,('xgb',XGBRegressor(learning_rate=0.08,
                                     n_estimators=3000, verbosity=1, max_depth=7))])
xgb.fit(train_X, train_y)
```

```
y_pred = xgb.predict(test_X)

rmse = np.sqrt(mean_squared_error(test_y,y_pred))
print('xgb RMSE:', rmse)
r2 = r2_score(test_y,y_pred)
print("R-squared:", r2)
```

```
xgb RMSE: 1.6110603602622964
R-squared: 0.9700761696735035
```

In [30]:
```
#icon+long
xgb = Pipeline([('prepocessor',preprocessor)
                ,('xgb',XGBRegressor(learning_rate=0.21,
                                     n_estimators=1600, verbosity=1, max_depth=6))])
xgb.fit(train_X, train_y)
y_pred = xgb.predict(test_X)

rmse = np.sqrt(mean_squared_error(test_y,y_pred))
print('xgb RMSE:', rmse)
r2 = r2_score(test_y,y_pred)
print("R-squared:", r2)
```

```
xgb RMSE: 1.6166667451737495
R-squared: 0.9698675413442925
```

In [29]:
```
xgb = Pipeline([('prepocessor',preprocessor)
                ,('xgb',XGBRegressor(learning_rate=0.21,
                                     n_estimators=2000, verbosity=1, max_depth=6))])
xgb.fit(train_X, train_y)
y_pred = xgb.predict(test_X)

rmse = np.sqrt(mean_squared_error(test_y,y_pred))
print('xgb RMSE:', rmse)
r2 = r2_score(test_y,y_pred)
print("R-squared:", r2)
```

```
xgb RMSE: 1.617117566422057
R-squared: 0.9698507336172549
```

In [39]:
```
xgb = Pipeline([('prepocessor',preprocessor)
                ,('xgb',XGBRegressor(learning_rate=0.4,
                                     n_estimators=1400, verbosity=1, max_depth=5))])
xgb.fit(train_X, train_y)
y_pred = xgb.predict(test_X)
```

```
rmse = np.sqrt(mean_squared_error(test_y,y_pred))
print('xgb RMSE:', rmse)
r2 = r2_score(test_y,y_pred)
print("R-squared:", r2)
```

```
xgb RMSE: 1.6288902740018953
R-squared: 0.9694101589918112
```

In [30]:
```
xgb = Pipeline([('prepocessor',preprocessor)
                ,('xgb',XGBRegressor(learning_rate=0.2,
                                     n_estimators=1500, verbosity=1, max_depth=6))])
xgb.fit(train_X, train_y)
y_pred = xgb.predict(test_X)

rmse = np.sqrt(mean_squared_error(test_y,y_pred))
print('xgb RMSE:', rmse)
r2 = r2_score(test_y,y_pred)
print("R-squared:", r2)
```

```
xgb RMSE: 1.6176050519744807
R-squared: 0.969832553681098
```

In [43]:
```
xgb = Pipeline([('prepocessor',preprocessor)
                ,('xgb',XGBRegressor(learning_rate=0.2,
                                     n_estimators=1500, verbosity=1, max_depth=7))])
xgb.fit(train_X, train_y)
y_pred = xgb.predict(test_X)

rmse = np.sqrt(mean_squared_error(test_y,y_pred))
print('xgb RMSE:', rmse)
r2 = r2_score(test_y,y_pred)
print("R-squared:", r2)
```

```
xgb RMSE: 1.6179007509430354
R-squared: 0.9698215234261124
```

In [36]:
```
xgb = Pipeline([('prepocessor',preprocessor)
                ,('xgb',XGBRegressor(learning_rate=0.21,
                                     n_estimators=1500, verbosity=1, max_depth=6))])
xgb.fit(train_X, train_y)
y_pred = xgb.predict(test_X)

rmse = np.sqrt(mean_squared_error(test_y,y_pred))
```

```
print('xgb RMSE:', rmse)
r2 = r2_score(test_y,y_pred)
print("R-squared:", r2)
```

```
xgb RMSE: 1.6168091928976431
R-squared: 0.9698622310485681
```

In [45]:
```
xgb = Pipeline([('prepocessor',preprocessor)
                ,('xgb',XGBRegressor(learning_rate=0.21,
                                     n_estimators=1499, verbosity=1, max_depth=6))])
xgb.fit(train_X, train_y)
y_pred = xgb.predict(test_X)

rmse = np.sqrt(mean_squared_error(test_y,y_pred))
print('xgb RMSE:', rmse)
r2 = r2_score(test_y,y_pred)
print("R-squared:", r2)
```

```
xgb RMSE: 1.6168267777108158
R-squared: 0.9698615754734751
```

In [20]:
```
gbr = Pipeline([('prepocessor',preprocessor),('gbr',GradientBoostingRegressor(n_estimators=110, learning_rate
gbr.fit(train_X, train_y)
y_pred = gbr.predict(test_X)

rmse = np.sqrt(mean_squared_error(test_y,y_pred))
print('gbr RMSE:', rmse)
r2 = r2_score(test_y,y_pred)
print("R-squared:", r2)
```

```
/Users/tangwenhua/opt/anaconda3/envs/dsc80/lib/python3.8/site-packages/sklearn/ensemble/_gb.py:437: DataConve
rsionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_s
amples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
gbr RMSE: 1.659661886397512
R-squared: 0.9682434876030509
```