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

In [2]:
```
# from google.colab import drive
# drive.mount('/content/drive')
```

In [3]:
```
ride = pd.read_csv("rideshare_kaggle.csv")
# ride = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/rideshare_kaggle.csv")
```

Modeling - Fare Price Prediction

In [4]:
```
#dropping rows with missing values
ride.dropna(axis=0,inplace=True)
```

In [5]:
```
import datetime

# Calling the fromtimestamp() function to
# extract datetime from the given timestamp
ride['timestamp']=pd.to_datetime(ride['timestamp'], unit='s')
timestamp = ride['timestamp']
timestamp
```

Out[5]:
```
0        2018-12-16 09:30:07.890000128
1        2018-11-27 02:00:23.676999936
2        2018-11-28 01:00:22.197999872
3        2018-11-30 04:53:02.749000192
4        2018-11-29 03:49:20.223000064
                     ...
693065   2018-12-01 23:53:06.000000000
693066   2018-12-01 23:53:06.000000000
693067   2018-12-01 23:53:06.000000000
693069   2018-12-01 23:53:06.000000000
693070   2018-12-01 23:53:06.000000000
Name: timestamp, Length: 637976, dtype: datetime64[ns]
```

In [6]:
```
ride.head(3)
```

Out[6]:

	id	timestamp	hour	day	month	datetime	timezone	source	dest
0	424553bb-7174-41ea-aeb4-fe06d4f4b9d7	2018-12-16 09:30:07.890000128	9	16	12	2018-12-16 09:30:07	America/New_York	Haymarket Square	
1	4bd23055-6827-41c6-b23b-3c491f24e74d	2018-11-27 02:00:23.676999936	2	27	11	2018-11-27 02:00:23	America/New_York	Haymarket Square	
2	981a3613-77af-4620-a42a-0c0866077d1e	2018-11-28 01:00:22.197999872	1	28	11	2018-11-28 01:00:22	America/New_York	Haymarket Square	

3 rows × 57 columns

In [7]:
```
# drop unnecessary column
ride = ride.drop(columns = ['id','timezone','timestamp', 'datetime', 'hour','month', '
```

In [ ]:
```
```

Observe the correlation among all numerical variables

In [8]:
```
numeric = ride._get_numeric_data()
numeric
```

Out[8]:

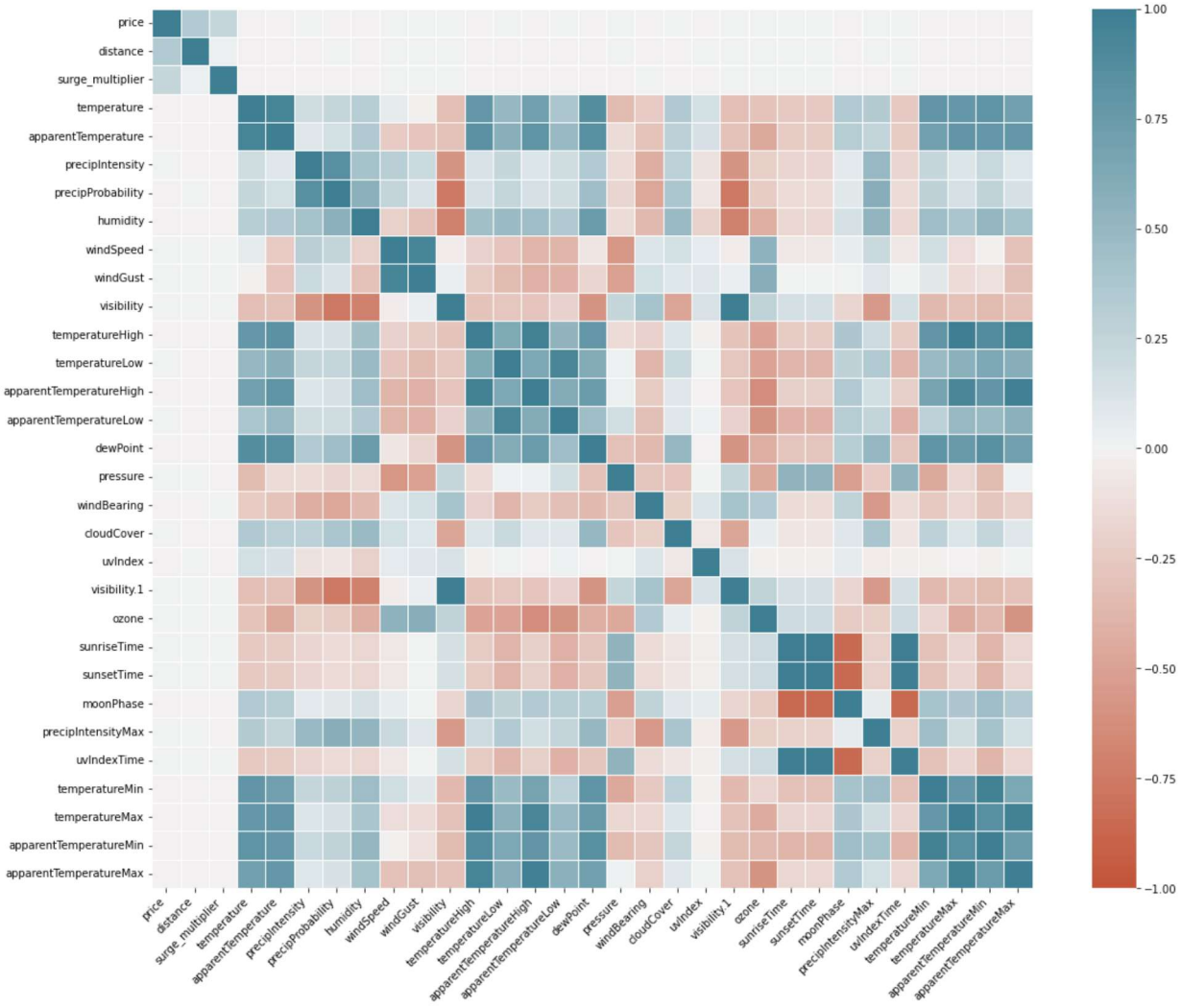
	price	distance	surge_multiplier	temperature	apparentTemperature	precipIntensity	precipP
0	5.0	0.44	1.0	42.34	37.12	0.0000	
1	11.0	0.44	1.0	43.58	37.35	0.1299	
2	7.0	0.44	1.0	38.33	32.93	0.0000	
3	26.0	0.44	1.0	34.38	29.63	0.0000	
4	9.0	0.44	1.0	37.44	30.88	0.0000	
...	...	...	...	...	...	...	
693065	9.5	1.00	1.0	37.05	37.05	0.0000	
693066	13.0	1.00	1.0	37.05	37.05	0.0000	
693067	9.5	1.00	1.0	37.05	37.05	0.0000	
693069	27.0	1.00	1.0	37.05	37.05	0.0000	
693070	10.0	1.00	1.0	37.05	37.05	0.0000	

637976 rows × 31 columns

In [9]:
```
numeric.columns
```

Out[9]:
```
Index(['price', 'distance', 'surge_multiplier', 'temperature',
       'apparentTemperature', 'precipIntensity', 'precipProbability',
       'humidity', 'windSpeed', 'windGust', 'visibility', 'temperatureHigh',
       'temperatureLow', 'apparentTemperatureHigh', 'apparentTemperatureLow',
       'dewPoint', 'pressure', 'windBearing', 'cloudCover', 'uvIndex',
       'visibility.1', 'ozone', 'sunriseTime', 'sunsetTime', 'moonPhase',
       'precipIntensityMax', 'uvIndexTime', 'temperatureMin', 'temperatureMax',
       'apparentTemperatureMin', 'apparentTemperatureMax'],
      dtype='object')
```

In [10]:
```
## how to improve the correlation map
data = numeric

corr = data.corr()
plt.subplots(figsize=(20,15))
ax = sns.heatmap(
    corr,
    vmin = -1, vmax = 1, center = 0,
    cmap = sns.diverging_palette(20, 220, n=200),
    square = True,
    linewidths=.5
)
ax.set_xticklabels(
    ax.get_xticklabels(),
    rotation = 45,
    horizontalalignment = 'right'
);
```



Encoding Categorical Variables

In [11]:
```
# get all categorical variables
obj = ride.dtypes == object
categorical_cols = ride.columns[obj]
print(categorical_cols)
```

```
Index(['source', 'destination', 'cab_type', 'name', 'icon'], dtype='object')
```

In [12]:
```
# Determine how many extra columns would be created
num_ohc_cols = (ride[categorical_cols]
                .apply(lambda x: x.nunique())
                .sort_values(ascending=False))
# No need to encode if there is only one value
small_num_ohc_cols = num_ohc_cols.loc[num_ohc_cols>1]

# Number of one-hot columns is one less than the number of categories
small_num_ohc_cols -= 1

small_num_ohc_cols.sum()
```

Out[12]:
```
40
```

In [13]:
```
%%time
```

```python
# encode categorical variables
from sklearn.preprocessing import OneHotEncoder, LabelEncoder

# The encoders
le = LabelEncoder()
ohc = OneHotEncoder()

for col in num_ohc_cols.index:

    # Integer encode the string categories
    dat = le.fit_transform(ride[col]).astype(int)

    # One hot encode the data--this returns a sparse array
    new_dat = ohc.fit_transform(dat.reshape(-1,1))

    # Create unique column names
    n_cols = new_dat.shape[1]
    col_names = ['_'.join([col, str(x)]) for x in range(n_cols)]

    # Create the new dataframe
    new_df = pd.DataFrame(new_dat.toarray(),
                          index = ride.index,
                          columns = col_names)
    value_colname_mapping = sorted(list(zip(ride[col].unique(), col_names)), key=lambd

    # Print original values and their one-hot encoded versions in the same order
    print(f"Original values and their one-hot encoded versions for {col}:")
    for original_value, one_hot_colname in value_colname_mapping:
        one_hot_value = np.zeros(n_cols)
        one_hot_value[col_names.index(one_hot_colname)] = 1
        print(f"{original_value}: {one_hot_value}")
    print("\n")

    # Append the new data to the dataframe
    ride = pd.concat([ride, new_df], axis=1)

    # Remove the original column from the dataframe
    ride = ride.drop(col, axis=1)
```

```
Original values and their one-hot encoded versions for source:
Back Bay: [0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
Beacon Hill: [0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
Boston University: [0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
Fenway: [0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
Financial District: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
Haymarket Square: [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
North End: [0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
North Station: [0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
Northeastern University: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
South Station: [0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
Theatre District: [0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
West End: [0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]


Original values and their one-hot encoded versions for destination:
Back Bay: [0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
Beacon Hill: [0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
Boston University: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
Fenway: [0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
Financial District: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
Haymarket Square: [0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
North End: [0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
North Station: [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
Northeastern University: [0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
South Station: [0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
Theatre District: [0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
West End: [0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]


Original values and their one-hot encoded versions for name:
Black: [0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
Black SUV: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
Lux: [0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
Lux Black: [0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
Lux Black XL: [0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
Lyft: [0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
Lyft XL: [0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
Shared: [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
UberPool: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
UberX: [0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
UberXL: [0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
WAV: [0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]


Original values and their one-hot encoded versions for icon:
 clear-day : [0. 0. 0. 0. 0. 1. 0.]
 clear-night : [0. 0. 1. 0. 0. 0. 0.]
 cloudy : [0. 0. 0. 1. 0. 0. 0.]
 fog : [0. 0. 0. 0. 1. 0. 0.]
 partly-cloudy-day : [0. 0. 0. 0. 0. 0. 1.]
 partly-cloudy-night : [1. 0. 0. 0. 0. 0. 0.]
 rain : [0. 1. 0. 0. 0. 0. 0.]


Original values and their one-hot encoded versions for cab_type:
Lyft: [1. 0.]
Uber: [0. 1.]
```

```
CPU times: total: 4.84 s
Wall time: 4.87 s
```

In [14]:
```
ride['timestamp'] = timestamp
```

In [15]:
```
# Get the date of every ride
ride["Date"] = ride["timestamp"].dt.date
del ride['timestamp']
```

Baseline Model - Take the price of a moving average of 1 day as baseline

In [16]:
```
# split the data
import statsmodels.api as sm

ride_train = ride.sample(frac=0.8, random_state=1)
ride_test = ride.drop(ride_train.index)
```

In [17]:
```
# calculate the average price of the day for training set
in_range_df = pd.pivot_table(ride_train, values=['price'],
                  index=['Date'],
                   aggfunc={'price':np.mean})
in_range_df['Date'] = in_range_df.index
in_range_df['Base_Price'] = in_range_df['price']
del in_range_df['price']
in_range_df
```

Out[17]:

Date	Date	Base_Price
2018-11-26	2018-11-26	16.617711
2018-11-27	2018-11-27	16.586300
2018-11-28	2018-11-28	16.522044
2018-11-29	2018-11-29	16.556943
2018-11-30	2018-11-30	16.437518
2018-12-01	2018-12-01	16.571723
2018-12-02	2018-12-02	16.525679
2018-12-03	2018-12-03	16.452130
2018-12-04	2018-12-04	16.700108
2018-12-09	2018-12-09	16.682334
2018-12-10	2018-12-10	16.463468
2018-12-13	2018-12-13	16.595583
2018-12-14	2018-12-14	16.545726
2018-12-15	2018-12-15	16.554902
2018-12-16	2018-12-16	16.578175
2018-12-17	2018-12-17	16.518330
2018-12-18	2018-12-18	16.549949

In [18]:
```
base_train = ride_train.set_index('Date').join(in_range_df.set_index('Date'), how='inr
base_train['Date'] = base_train.index
del base_train['Date']
# del base_train['timestamp']
# base_train.head(3)
```

In [19]:
```
# calculate the average price of the day for testing set
in_range_df_1 = pd.pivot_table(ride_test, values=['price'],
                 index=['Date'],
                  aggfunc={'price':np.mean})
in_range_df_1['Date'] = in_range_df_1.index
in_range_df_1['Base_Price'] = in_range_df_1['price']
del in_range_df_1['price']
in_range_df_1
```

Out[19]:

	Date	Base_Price
Date		
2018-11-26	2018-11-26	16.179391
2018-11-27	2018-11-27	16.605152
2018-11-28	2018-11-28	16.542629
2018-11-29	2018-11-29	16.595253
2018-11-30	2018-11-30	16.342112
2018-12-01	2018-12-01	16.575284
2018-12-02	2018-12-02	16.642462
2018-12-03	2018-12-03	16.514854
2018-12-04	2018-12-04	16.470301
2018-12-09	2018-12-09	17.271186
2018-12-10	2018-12-10	16.235967
2018-12-13	2018-12-13	16.566172
2018-12-14	2018-12-14	16.616540
2018-12-15	2018-12-15	16.583616
2018-12-16	2018-12-16	16.621050
2018-12-17	2018-12-17	16.412860
2018-12-18	2018-12-18	16.645758

In [20]:
```
base_test = ride_test.set_index('Date').join(in_range_df_1.set_index('Date'), how='inr
base_test['Date'] = base_test.index
del base_test['Date']
base_test.head(3)
```

Out[20]:

	price	distance	surge_multiplier	temperature	apparentTemperature	precipIntensity	precipPrc
Date							
2018-11-26	13.5	2.36	1.0	41.30	41.30	0.0000	
2018-11-26	12.5	4.43	1.0	41.99	41.99	0.0000	
2018-11-26	32.5	3.18	1.0	45.80	43.94	0.0024	

3 rows × 77 columns

In [21]:
```
X_base_train = base_train.drop(['Base_Price'], axis=1) # was ride_train[features]
y_base_train = base_train['Base_Price']

X_base_test = base_test.drop(['Base_Price'], axis=1) #col was ride_test[features]
```

```
y_base_test = base_test['Base_Price']

# We must add an intercept as the standard model doesn't automatically fit one
X_base_train = sm.add_constant(X_base_train)
X_base_test = sm.add_constant(X_base_test) #added this line
# fit the data to the model
model = sm.OLS(y_base_train, X_base_train).fit()

print(model.summary())
```

OLS Regression Results

Dep. Variable:	Base_Price	R-squared:	0.657
Model:	OLS	Adj. R-squared:	0.657
Method:	Least Squares	F-statistic:	1.458e+04
Date:	Thu, 27 Apr 2023	Prob (F-statistic):	0.00
Time:	22:03:15	Log-Likelihood:	1.0625e+06
No. Observations:	510381	AIC:	-2.125e+06
Df Residuals:	510313	BIC:	-2.124e+06
Df Model:	67		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	124.2400	1.276	97.397	0.000	121.740	126.740
price	1.949e-05	1.69e-05	1.151	0.250	-1.37e-05	5.27e-05
distance	-0.0001	6.89e-05	-1.557	0.119	-0.000	2.77e-05
surge_multiplier	-0.0007	0.001	-1.351	0.177	-0.002	0.000
temperature	0.0055	0.000	33.988	0.000	0.005	0.006
apparentTemperature	-0.0100	5.6e-05	-178.475	0.000	-0.010	-0.010
precipIntensity	-0.1699	0.003	-49.741	0.000	-0.177	-0.163
precipProbability	0.0099	0.001	14.240	0.000	0.009	0.011
humidity	-0.2072	0.005	-40.821	0.000	-0.217	-0.197
windSpeed	-0.0034	6.64e-05	-51.295	0.000	-0.004	-0.003
windGust	-0.0002	3.05e-05	-5.402	0.000	-0.000	-0.000
visibility	0.0006	2.14e-05	27.703	0.000	0.001	0.001
temperatureHigh	-0.3120	0.001	-211.128	0.000	-0.315	-0.309
temperatureLow	0.0016	4.69e-05	34.367	0.000	0.002	0.002
apparentTemperatureHigh	0.1796	0.001	208.796	0.000	0.178	0.181
apparentTemperatureLow	-0.0025	3.89e-05	-63.078	0.000	-0.003	-0.002
dewPoint	0.0055	0.000	39.115	0.000	0.005	0.006
pressure	0.0032	1.26e-05	255.867	0.000	0.003	0.003
windBearing	-5.813e-05	7.82e-07	-74.352	0.000	-5.97e-05	-5.66e-05
cloudCover	-0.0033	0.000	-10.019	0.000	-0.004	-0.003
uvIndex	-0.0016	0.000	-14.385	0.000	-0.002	-0.001
visibility.1	0.0006	2.14e-05	29.565	0.000	0.001	

0.001
ozone 0.0016 4.73e-06 328.920 0.000 0.002
0.002
sunriseTime 8.448e-05 1.99e-06 42.498 0.000 8.06e-05 8.
84e-05
sunsetTime -9.446e-05 2e-06 -47.136 0.000 -9.84e-05 -9.
05e-05
moonPhase -0.1132 0.001 -154.836 0.000 -0.115
-0.112
precipIntensityMax 0.3227 0.002 184.769 0.000 0.319
0.326
uvIndexTime 9.847e-06 4.7e-08 209.568 0.000 9.76e-06 9.
94e-06
temperatureMin 0.0025 5.19e-05 48.464 0.000 0.002
0.003
temperatureMax 0.3046 0.001 206.958 0.000 0.302
0.308
apparentTemperatureMin -0.0029 6.08e-05 -48.312 0.000 -0.003
-0.003
apparentTemperatureMax -0.1697 0.001 -198.172 0.000 -0.171
-0.168
source_0 10.3787 0.107 97.398 0.000 10.170
10.588
source_1 10.3787 0.107 97.397 0.000 10.170
10.588
source_2 10.3788 0.107 97.399 0.000 10.170
10.588
source_3 10.3789 0.107 97.399 0.000 10.170
10.588
source_4 10.3789 0.107 97.399 0.000 10.170
10.588
source_5 10.3784 0.107 97.395 0.000 10.170
10.587
source_6 10.3787 0.107 97.397 0.000 10.170
10.588
source_7 10.3787 0.107 97.397 0.000 10.170
10.588
source_8 10.3787 0.107 97.397 0.000 10.170
10.588
source_9 10.3787 0.107 97.397 0.000 10.170
10.588
source_10 10.3783 0.107 97.393 0.000 10.169
10.587
source_11 10.3785 0.107 97.396 0.000 10.170
10.587
destination_0 10.3783 0.107 97.394 0.000 10.169
10.587
destination_1 10.3784 0.107 97.394 0.000 10.170
10.587
destination_2 10.3790 0.107 97.400 0.000 10.170
10.588
destination_3 10.3789 0.107 97.399 0.000 10.170
10.588
destination_4 10.3788 0.107 97.398 0.000 10.170
10.588
destination_5 10.3783 0.107 97.393 0.000 10.169
10.587
destination_6 10.3785 0.107 97.395 0.000 10.170
10.587
destination_7 10.3789 0.107 97.399 0.000 10.170

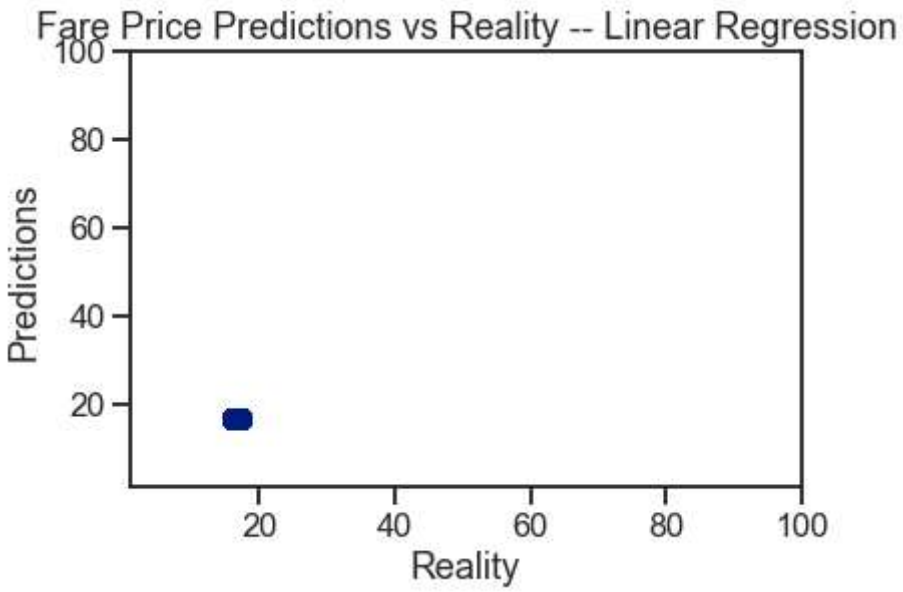
```
10.588
destination_8              10.3786      0.107     97.396      0.000      10.170
10.587
destination_9              10.3786      0.107     97.397      0.000      10.170
10.587
destination_10             10.3788      0.107     97.398      0.000      10.170
10.588
destination_11             10.3791      0.107     97.401      0.000      10.170
10.588
name_0                     10.3787      0.107     97.397      0.000      10.170
10.588
name_1                     10.3785      0.107     97.394      0.000      10.170
10.587
name_2                     10.3787      0.107     97.397      0.000      10.170
10.588
name_3                     10.3786      0.107     97.396      0.000      10.170
10.587
name_4                     10.3786      0.107     97.396      0.000      10.170
10.588
name_5                     10.3786      0.107     97.397      0.000      10.170
10.587
name_6                     10.3787      0.107     97.398      0.000      10.170
10.588
name_7                     10.3788      0.107     97.398      0.000      10.170
10.588
name_8                     10.3787      0.107     97.398      0.000      10.170
10.588
name_9                     10.3786      0.107     97.396      0.000      10.170
10.587
name_10                    10.3787      0.107     97.398      0.000      10.170
10.588
name_11                    10.3787      0.107     97.397      0.000      10.170
10.588
icon_0                     17.7796      0.183     97.322      0.000      17.421
18.138
icon_1                     17.7661      0.183     97.272      0.000      17.408
18.124
icon_2                     17.7927      0.183     97.392      0.000      17.435
18.151
icon_3                     17.7935      0.183     97.423      0.000      17.435
18.151
icon_4                     17.7986      0.183     97.430      0.000      17.441
18.157
icon_5                     17.8088      0.183     97.482      0.000      17.451
18.167
icon_6                     17.8046      0.183     97.459      0.000      17.447
18.163
cab_type_0                 62.2721      0.639     97.397      0.000      61.019
63.525
cab_type_1                 62.2719      0.639     97.397      0.000      61.019
63.525
==============================================================================
Omnibus:                   139895.330   Durbin-Watson:                   0.971
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           949877.167
Skew:                           1.148   Prob(JB):                         0.00
Kurtosis:                       9.277   Cond. No.                     4.84e+24
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly spec
```

```
ified.
[2] The smallest eigenvalue is 1.56e-25. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

In [22]:
```python
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

LR = LinearRegression()
LR = LR.fit(X_base_train, y_base_train)
y_train_pred = LR.predict(X_base_train)
y_test_pred = LR.predict(X_base_test)

# MSE
print('train mse: ', np.sqrt(mean_squared_error(y_base_train, y_train_pred)))
print('test mse: ',  np.sqrt(mean_squared_error(y_base_test, y_test_pred)))

# R2 score
model1 = LinearRegression().fit(X_base_train, y_base_train)
R_squared = model1.score(X_base_train, y_base_train)
print('R_squared:', R_squared)
```

```
train mse:  0.030175315381933382
test mse:  0.12149411147468951
R_squared: 0.6568656755045368
```

In [23]:
```python
# Plot test dataset Prediction
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

sns.set_context('talk')
sns.set_style('ticks')
sns.set_palette('dark')

ax = plt.axes()
# use y_test, y_test_pred
ax.scatter(y_base_test, y_test_pred, alpha=.5)
ax.set_xlim(1, 100)
ax.set_ylim(1, 100)
ax.set(xlabel='Reality',
       ylabel='Predictions',
       title='Fare Price Predictions vs Reality -- Linear Regression');
```



Linear Regression Model

In [24]:
```
# Choose the features to be used

del ride_train['Date']
del ride_test['Date']

X_train = ride_train.drop(['price'], axis=1) # was ride_train[features]
y_train = ride_train['price']

X_test = ride_test.drop(['price'], axis=1) #col was ride_test[features]
y_test = ride_test['price']

# We must add an intercept as the standard model doesn't automatically fit one
X_train = sm.add_constant(X_train)
X_test = sm.add_constant(X_test) #added this line
# fit the data to the model
model = sm.OLS(y_train, X_train).fit()

print(model.summary())
```

OLS Regression Results

Dep. Variable:	price	R-squared:	0.929
Model:	OLS	Adj. R-squared:	0.929
Method:	Least Squares	F-statistic:	1.005e+05
Date:	Thu, 27 Apr 2023	Prob (F-statistic):	0.00
Time:	22:03:21	Log-Likelihood:	-1.1905e+06
No. Observations:	510381	AIC:	2.381e+06
Df Residuals:	510314	BIC:	2.382e+06
Df Model:	66		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	243.5471	105.392	2.311	0.021	36.982	450.112
distance	2.8837	0.004	719.381	0.000	2.876	2.892
surge_multiplier	18.4984	0.037	493.481	0.000	18.425	18.572
temperature	0.0079	0.013	0.586	0.558	-0.018	0.034
apparentTemperature	-0.0031	0.005	-0.665	0.506	-0.012	0.006
precipIntensity	-0.3121	0.282	-1.105	0.269	-0.865	0.241
precipProbability	0.0094	0.058	0.163	0.871	-0.103	0.122
humidity	0.0396	0.419	0.094	0.925	-0.782	0.861
windSpeed	-0.0087	0.005	-1.592	0.111	-0.019	0.002
windGust	0.0044	0.003	1.736	0.083	-0.001	0.009
visibility	-0.0012	0.002	-0.687	0.492	-0.005	0.002
temperatureHigh	0.0991	0.122	0.812	0.417	-0.140	0.338
temperatureLow	0.0031	0.004	0.789	0.430	-0.005	0.011
apparentTemperatureHigh	-0.0671	0.071	-0.943	0.345	-0.206	0.072
apparentTemperatureLow	-0.0024	0.003	-0.756	0.450	-0.009	0.004
dewPoint	-0.0025	0.012	-0.214	0.831	-0.025	0.020
pressure	0.0003	0.001	0.334	0.738	-0.002	0.002
windBearing	0.0001	6.46e-05	1.767	0.077	-1.25e-05	0.000
cloudCover	-0.0393	0.027	-1.438	0.150	-0.093	0.014
uvIndex	-0.0091	0.009	-0.962	0.336	-0.028	0.009
visibility.1	-0.0012	0.002	-0.681	0.496	-0.005	0.002
ozone	-0.0002	0.000	-0.401	0.688	-0.001	

0.001
sunriseTime 0.0004 0.000 2.347 0.019 6.36e-05
0.001
sunsetTime -0.0004 0.000 -2.334 0.020 -0.001 -
6.2e-05
moonPhase -0.0911 0.060 -1.509 0.131 -0.209
0.027
precipIntensityMax 0.0236 0.144 0.164 0.870 -0.259
0.306
uvIndexTime 7.454e-07 3.88e-06 0.192 0.848 -6.86e-06 8.
35e-06
temperatureMin 0.0058 0.004 1.346 0.178 -0.003
0.014
temperatureMax -0.1095 0.122 -0.900 0.368 -0.348
0.129
apparentTemperatureMin -0.0003 0.005 -0.059 0.953 -0.010
0.010
apparentTemperatureMax 0.0705 0.071 0.997 0.319 -0.068
0.209
source_0 20.2005 8.804 2.294 0.022 2.945
37.456
source_1 19.8967 8.804 2.260 0.024 2.641
37.153
source_2 20.0188 8.804 2.274 0.023 2.763
37.275
source_3 20.2104 8.804 2.296 0.022 2.954
37.466
source_4 20.5508 8.804 2.334 0.020 3.295
37.807
source_5 20.6680 8.804 2.348 0.019 3.412
37.924
source_6 20.8493 8.804 2.368 0.018 3.593
38.105
source_7 20.2853 8.804 2.304 0.021 3.029
37.541
source_8 19.9987 8.804 2.272 0.023 2.743
37.255
source_9 20.4617 8.804 2.324 0.020 3.206
37.718
source_10 20.7201 8.804 2.353 0.019 3.464
37.976
source_11 20.2826 8.804 2.304 0.021 3.027
37.538
destination_0 20.2826 8.804 2.304 0.021 3.027
37.538
destination_1 19.9751 8.804 2.269 0.023 2.719
37.231
destination_2 20.3308 8.804 2.309 0.021 3.075
37.587
destination_3 20.0152 8.804 2.273 0.023 2.759
37.271
destination_4 20.6881 8.804 2.350 0.019 3.432
37.944
destination_5 20.5513 8.804 2.334 0.020 3.295
37.807
destination_6 20.3955 8.804 2.317 0.021 3.140
37.651
destination_7 20.4906 8.804 2.327 0.020 3.235
37.746
destination_8 20.3378 8.804 2.310 0.021 3.082

```
37.594
destination_9              20.3053      8.804      2.306      0.021       3.049
37.561
destination_10             20.5158      8.804      2.330      0.020       3.260
37.772
destination_11             20.2547      8.804      2.301      0.021       2.999
37.510
name_0                     25.0136      8.804      2.841      0.004       7.758
42.269
name_1                     34.7689      8.804      3.949      0.000      17.513
52.025
name_2                     20.7311      8.804      2.355      0.019       3.475
37.987
name_3                     26.0084      8.804      2.954      0.003       8.753
43.264
name_4                     35.2843      8.804      4.008      0.000      18.028
52.540
name_5                     12.5483      8.804      1.425      0.154      -4.708
29.804
name_6                     18.2542      8.804      2.073      0.038       0.998
35.510
name_7                      9.6703      8.804      1.098      0.272      -7.586
26.926
name_8                     13.2299      8.804      1.503      0.133      -4.026
30.486
name_9                     14.2372      8.804      1.617      0.106      -3.019
31.493
name_10                    20.1636      8.804      2.290      0.022       2.908
37.419
name_11                    14.2330      8.804      1.617      0.106      -3.023
31.489
icon_0                     34.8504     15.094      2.309      0.021       5.267
64.434
icon_1                     34.8587     15.090      2.310      0.021       5.282
64.435
icon_2                     34.8897     15.094      2.311      0.021       5.306
64.474
icon_3                     34.8514     15.090      2.310      0.021       5.275
64.428
icon_4                     34.8870     15.093      2.311      0.021       5.304
64.470
icon_5                     34.8994     15.094      2.312      0.021       5.316
64.483
icon_6                     34.9062     15.094      2.313      0.021       5.322
64.490
cab_type_0                122.4967     52.825      2.319      0.020      18.962     2
26.032
cab_type_1                121.6462     52.825      2.303      0.021      18.111     2
25.181
==============================================================================
Omnibus:                   196527.429   Durbin-Watson:                   1.996
Prob(Omnibus):                  0.000   Jarque-Bera (JB):          3697885.963
Skew:                           1.379   Prob(JB):                         0.00
Kurtosis:                      15.895   Cond. No.                     2.75e+24
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly spec
ified.
```

```
[2] The smallest eigenvalue is 4.81e-25. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

In [25]:
```
X_train
```

Out[25]:

	const	distance	surge_multiplier	temperature	apparentTemperature	precipIntensity	precipP
658311	1.0	2.16	1.0	41.34	34.55	0.0000	
547209	1.0	2.25	1.0	51.84	51.84	0.0000	
225076	1.0	1.56	1.0	36.95	36.95	0.0000	
360104	1.0	0.74	1.0	41.35	35.62	0.0000	
635115	1.0	3.14	1.0	37.71	31.92	0.0000	
...	...	...	...	...	...	...	
450255	1.0	3.58	1.0	40.77	35.14	0.0000	
299568	1.0	3.39	1.0	47.87	47.87	0.0000	
642890	1.0	3.14	1.0	40.38	35.18	0.0000	
608760	1.0	2.87	1.0	31.71	22.64	0.0025	
278771	1.0	3.07	1.0	24.71	12.26	0.0000	

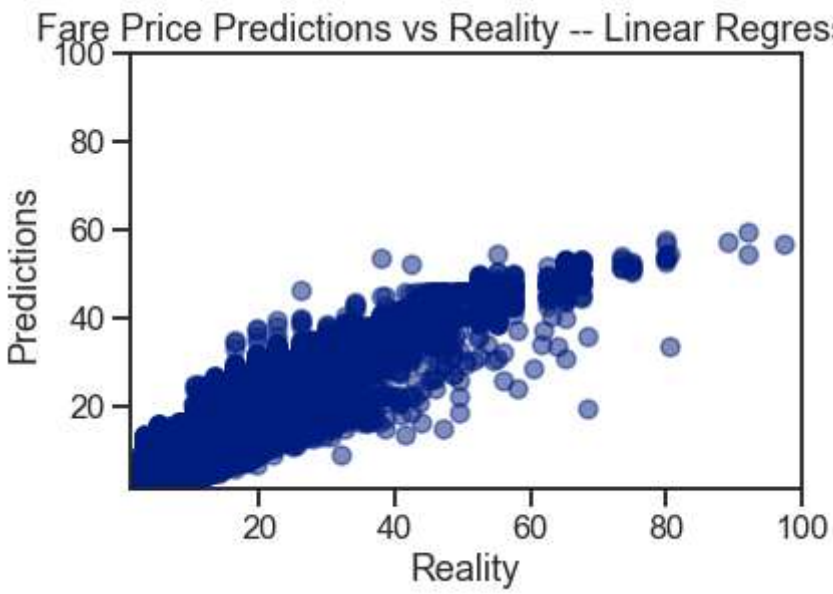
510381 rows × 76 columns

In [26]:
```
# Linear Regression
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

LR = LinearRegression()
LR = LR.fit(X_train, y_train)
y_train_pred = LR.predict(X_train)
y_test_pred = LR.predict(X_test)

# MSE
print('train mse: ', np.sqrt(mean_squared_error(y_train, y_train_pred)))
print('test mse: ',  np.sqrt(mean_squared_error(y_test, y_test_pred)))

# R2 score
from sklearn.metrics import r2_score
print('r2 score: ', r2_score(y_test, y_test_pred))
```

```
train mse:  2.493138522118031
test mse:  2.484092998159326
r2 score:  0.9286870041997767
```

In [27]:
```
# Plot test dataset Prediction
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

sns.set_context('talk')
sns.set_style('ticks')
sns.set_palette('dark')
```

```
ax = plt.axes()
# use y_test, y_test_pred
ax.scatter(y_test, y_test_pred, alpha=.5)
ax.set_xlim(1, 100)
ax.set_ylim(1, 100)
ax.set(xlabel='Reality',
       ylabel='Predictions',
       title='Fare Price Predictions vs Reality -- Linear Regression');
```



In [28]:
```
# Cross Validation
from sklearn.model_selection  import KFold, GridSearchCV
from sklearn.pipeline import Pipeline
```

In [29]:
```
from sklearn.model_selection import cross_val_score
lr = LinearRegression()
scores = cross_val_score(lr, X_train, y_train, cv=5)

print("%0.2f accuracy" % (scores.mean()))
```

```
0.93 accuracy
```

In [30]:
```
X_train
```

Out[30]:

	const	distance	surge_multiplier	temperature	apparentTemperature	precipIntensity	precipP
658311	1.0	2.16	1.0	41.34	34.55	0.0000	
547209	1.0	2.25	1.0	51.84	51.84	0.0000	
225076	1.0	1.56	1.0	36.95	36.95	0.0000	
360104	1.0	0.74	1.0	41.35	35.62	0.0000	
635115	1.0	3.14	1.0	37.71	31.92	0.0000	
...	...	...	...	...	...	...	
450255	1.0	3.58	1.0	40.77	35.14	0.0000	
299568	1.0	3.39	1.0	47.87	47.87	0.0000	
642890	1.0	3.14	1.0	40.38	35.18	0.0000	
608760	1.0	2.87	1.0	31.71	22.64	0.0025	
278771	1.0	3.07	1.0	24.71	12.26	0.0000	

510381 rows × 76 columns

In [31]:
```
X_train.columns
```

Out[31]:
```
Index(['const', 'distance', 'surge_multiplier', 'temperature',
       'apparentTemperature', 'precipIntensity', 'precipProbability',
       'humidity', 'windSpeed', 'windGust', 'visibility', 'temperatureHigh',
       'temperatureLow', 'apparentTemperatureHigh', 'apparentTemperatureLow',
       'dewPoint', 'pressure', 'windBearing', 'cloudCover', 'uvIndex',
       'visibility.1', 'ozone', 'sunriseTime', 'sunsetTime', 'moonPhase',
       'precipIntensityMax', 'uvIndexTime', 'temperatureMin', 'temperatureMax',
       'apparentTemperatureMin', 'apparentTemperatureMax', 'source_0',
       'source_1', 'source_2', 'source_3', 'source_4', 'source_5', 'source_6',
       'source_7', 'source_8', 'source_9', 'source_10', 'source_11',
       'destination_0', 'destination_1', 'destination_2', 'destination_3',
       'destination_4', 'destination_5', 'destination_6', 'destination_7',
       'destination_8', 'destination_9', 'destination_10', 'destination_11',
       'name_0', 'name_1', 'name_2', 'name_3', 'name_4', 'name_5', 'name_6',
       'name_7', 'name_8', 'name_9', 'name_10', 'name_11', 'icon_0', 'icon_1',
       'icon_2', 'icon_3', 'icon_4', 'icon_5', 'icon_6', 'cab_type_0',
       'cab_type_1'],
      dtype='object')
```

In [32]:
```
X_train = X_train.drop(columns = ['const','temperature',
       'apparentTemperature', 'precipIntensity', 'precipProbability',
       'humidity', 'windSpeed', 'windGust', 'visibility', 'temperatureHigh',
       'temperatureLow', 'apparentTemperatureHigh', 'apparentTemperatureLow',
       'dewPoint', 'pressure', 'windBearing', 'cloudCover', 'uvIndex',
       'visibility.1', 'ozone', 'sunriseTime', 'sunsetTime', 'moonPhase',
       'precipIntensityMax', 'uvIndexTime', 'temperatureMin', 'temperatureMax',
       'apparentTemperatureMin', 'apparentTemperatureMax','source_0',
       'source_1', 'source_2', 'source_3', 'source_4', 'source_5', 'source_6',
       'source_7', 'source_8', 'source_9', 'source_10', 'source_11',
       'destination_0', 'destination_1', 'destination_2', 'destination_3',
       'destination_4', 'destination_5', 'destination_6', 'destination_7',
       'destination_8', 'destination_9', 'destination_10', 'destination_11', 'icon_0',
```

```
       'icon_2', 'icon_3', 'icon_4', 'icon_5', 'icon_6', 'cab_type_0',
       'cab_type_1'])
X_test = X_test.drop(columns =['const','temperature',
       'apparentTemperature', 'precipIntensity', 'precipProbability',
       'humidity', 'windSpeed', 'windGust', 'visibility', 'temperatureHigh',
       'temperatureLow', 'apparentTemperatureHigh', 'apparentTemperatureLow',
       'dewPoint', 'pressure', 'windBearing', 'cloudCover', 'uvIndex',
       'visibility.1', 'ozone', 'sunriseTime', 'sunsetTime', 'moonPhase',
       'precipIntensityMax', 'uvIndexTime', 'temperatureMin', 'temperatureMax',
       'apparentTemperatureMin', 'apparentTemperatureMax','source_0',
       'source_1', 'source_2', 'source_3', 'source_4', 'source_5', 'source_6',
       'source_7', 'source_8', 'source_9', 'source_10', 'source_11',
       'destination_0', 'destination_1', 'destination_2', 'destination_3',
       'destination_4', 'destination_5', 'destination_6', 'destination_7',
       'destination_8', 'destination_9', 'destination_10', 'destination_11', 'icon_0',
       'icon_2', 'icon_3', 'icon_4', 'icon_5', 'icon_6', 'cab_type_0',
       'cab_type_1'] )
```

In [33]:
```
X_train
```

Out[33]:

	distance	surge_multiplier	name_0	name_1	name_2	name_3	name_4	name_5	name_6	na
658311	2.16	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	
547209	2.25	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	
225076	1.56	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
360104	0.74	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
635115	3.14	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	...	...
450255	3.58	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
299568	3.39	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
642890	3.14	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
608760	2.87	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
278771	3.07	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

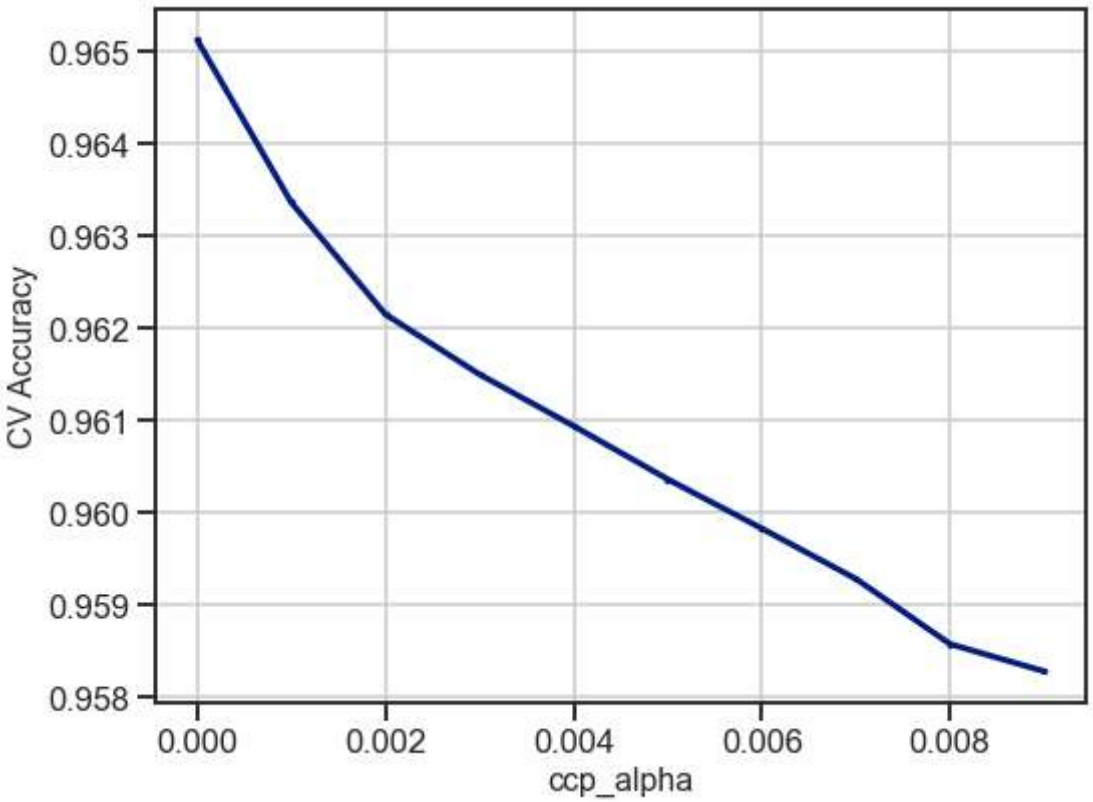
510381 rows × 14 columns

Decision tree regressor

In [34]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeRegressor

grid_values = {'ccp_alpha': np.arange(0, 0.01, 0.001).tolist(),
               'min_samples_leaf': [5],
                'min_samples_split': [20],
                'max_depth': [30],
                'random_state': [88]}
```

```
dtc = DecisionTreeRegressor(random_state=88)
dtc_cv = GridSearchCV(dtc, param_grid=grid_values, cv=2).fit(X_train, y_train)
```

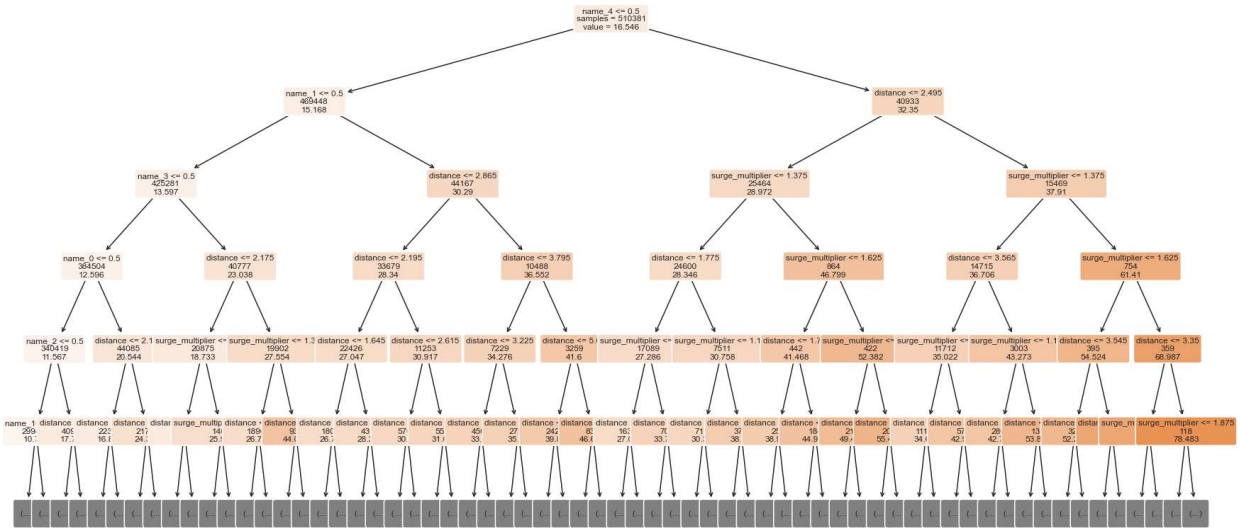
In [35]:
```
import matplotlib.pyplot as plt

ccp_alpha = dtc_cv.cv_results_['param_ccp_alpha'].data
ACC_scores = dtc_cv.cv_results_['mean_test_score']

plt.figure(figsize=(8, 6))
plt.xlabel('ccp_alpha', fontsize=16)
plt.ylabel('CV Accuracy', fontsize=16)
plt.scatter(ccp_alpha, ACC_scores, s=3)
plt.plot(ccp_alpha, ACC_scores, linewidth=3)
plt.grid(True, which='both')

plt.tight_layout()
plt.show()

print('Best ccp_alpha', dtc_cv.best_params_)
```




```
Best ccp_alpha {'ccp_alpha': 0.0, 'max_depth': 30, 'min_samples_leaf': 5, 'min_sample
s_split': 20, 'random_state': 88}
```

In [36]:
```
from sklearn.tree import plot_tree
plt.figure(figsize=(30,15))
plot_tree(dtc_cv.best_estimator_,
          feature_names= X_train.columns,
          class_names=['0','1'],
          filled=True,
          impurity=False,
          rounded=True,
          fontsize=12,
          max_depth = 5,
```

```
            label='root')
plt.show()
```



In [37]:
```
from sklearn.metrics import r2_score

print('CV R2:', round(dtc_cv.best_score_, 5))
print('OSR2:', round(r2_score(y_test, dtc_cv.predict(X_test)), 5))
```

```
CV R2: 0.96513
OSR2: 0.96485
```

XG BOOST

In [38]:
```
pip install xgboost
```

```
Requirement already satisfied: xgboost in c:\users\bhaws\anaconda3\lib\site-packages
(1.7.5)
Requirement already satisfied: numpy in c:\users\bhaws\anaconda3\lib\site-packages (f
rom xgboost) (1.21.5)
Requirement already satisfied: scipy in c:\users\bhaws\anaconda3\lib\site-packages (f
rom xgboost) (1.7.3)
Note: you may need to restart the kernel to use updated packages.
```

In [39]:
```
import xgboost as xg
from sklearn.metrics import mean_squared_error as MSE
```

In [40]:
```
#This is to deal with the error assocoated with uniquness of our features(duplicate co
X_train = X_train.loc[:,~X_train.columns.duplicated()].copy()
X_test = X_test.loc[:,~X_test.columns.duplicated()].copy()
```

In [41]:
```
X_train
```

Out[41]:

	distance	surge_multiplier	name_0	name_1	name_2	name_3	name_4	name_5	name_6	na
658311	2.16	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	
547209	2.25	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	
225076	1.56	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
360104	0.74	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
635115	3.14	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	...	...
450255	3.58	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
299568	3.39	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
642890	3.14	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
608760	2.87	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
278771	3.07	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

510381 rows × 14 columns

In [42]:

```
# Various hyper-parameters to tune
xgb1 = xg.XGBRegressor()
parameters = {'nthread':[-1], #when use hyperthread, xgboost may become slower
              'objective':['reg:squarederror'],
              'learning_rate': [.03, 0.05, .07], #so called `eta` value
              'max_depth': [5, 6, 7],
              'min_child_weight': [4],
              'silent': [1],
              'subsample': [0.7],
              'colsample_bytree': [0.7],
              'n_estimators': [100]}

xgb_grid = GridSearchCV(xgb1,
                        parameters,
                        cv = 2,
                        n_jobs = -1,
                        verbose=False)

xgb_grid.fit(X_train,
         y_train)
```

```
[22:04:36] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-group
-i-07593ffd91cd9da33-1\xgboost\xgboost-ci-windows\src\learner.cc:767:
Parameters: { "silent" } are not used.
```

Out[42]:
```
GridSearchCV(cv=2,
             estimator=XGBRegressor(base_score=None, booster=None,
                                    callbacks=None, colsample_bylevel=None,
                                    colsample_bynode=None,
                                    colsample_bytree=None,
                                    early_stopping_rounds=None,
                                    enable_categorical=False, eval_metric=None,
                                    feature_types=None, gamma=None, gpu_id=None,
                                    grow_policy=None, importance_type=None,
                                    interaction_constraints=None,
                                    learning_rate=None, m...
                                    monotone_constraints=None, n_estimators=100,
                                    n_jobs=None, num_parallel_tree=None,
                                    predictor=None, random_state=None, ...),
             n_jobs=-1,
             param_grid={'colsample_bytree': [0.7],
                         'learning_rate': [0.03, 0.05, 0.07],
                         'max_depth': [5, 6, 7], 'min_child_weight': [4],
                         'n_estimators': [100], 'nthread': [-1],
                         'objective': ['reg:squarederror'], 'silent': [1],
                         'subsample': [0.7]},
             verbose=False)
```

In [43]:
```
print('CV R2:', xgb_grid.best_score_, 5)
XG_OSR2 = r2_score(y_test, xgb_grid.predict(X_test))
print('OSR2:', round(XG_OSR2, 5))
print('Best parameters', xgb_grid.best_params_)
```

```
CV R2: 0.9641773889345275 5
OSR2: 0.96344
Best parameters {'colsample_bytree': 0.7, 'learning_rate': 0.07, 'max_depth': 7, 'min
_child_weight': 4, 'n_estimators': 100, 'nthread': -1, 'objective': 'reg:squarederro
r', 'silent': 1, 'subsample': 0.7}
```

In [ ]:
```
```

In [44]:
```
xgb1.fit(X_train,
         y_train)

model_file = 'xgb_model.bin'
xgb1.save_model(model_file)
```

In [ ]:
```
import sys
import xgboost as xgb
import pandas as pd
from PyQt5.QtWidgets import QApplication, QWidget, QLabel, QLineEdit, QPushButton, QVB
from PyQt5.QtGui import QFont, QIcon

# Define the input variables
input_vars = ['Distance', 'Surge', 'Lyft Shared', 'Lyft Lux', 'Lyft', 'Lyft Lux Black

# Load the trained XGBoost model
xgb_model = xgb.Booster()
xgb_model.load_model(model_file)

class MainWindow(QWidget):
    def __init__(self):
        super().__init__()
```

```
        # Set custom font
        font = QFont("Arial", 16)

        # Create the GUI elements
        self.labels = []
        self.line_edits = []
        self.button = QPushButton('Predict')
        self.result_label = QLabel()
        self.ride_type_combobox = QComboBox()

        # Set the style sheet for the window
        self.setStyleSheet('''
            QWidget {
                background-color: #FFF;
                color: #444;
            }

            QLabel {
                font-size: 16px;
            }

            QLineEdit {
                font-size: 16px;
                padding: 5px;
                border: 2px solid #DDD;
                border-radius: 5px;
            }

            QPushButton {
                font-size: 16px;
                padding: 5px;
                border: none;
                border-radius: 5px;
                background-color: #5FCCF5;
                color: #FFF;
            }

            QPushButton:hover {
                background-color: #3FA6C5;
            }

            QLabel#result_label {
                font-size: 20px;
                color: #444;
                margin-top: 20px;
            }

            QComboBox {
                font-size: 16px;
                padding: 5px;
                border: 2px solid #DDD;
                border-radius: 5px;
            }
        ''')

        # Create the input elements
        for var in input_vars:
            label = QLabel(f'{var}:')
            label.setFont(font)
            self.labels.append(label)
```

```
        self.distance_edit = QLineEdit()
        self.distance_edit.setFont(font)
        self.surge_edit = QLineEdit()
        self.surge_edit.setFont(font)
        self.line_edits = [self.distance_edit, self.surge_edit]

        for var in input_vars[2:]:
            self.ride_type_combobox.addItem(var)

        # Create a form layout
        form_layout = QFormLayout()

        # Add the input elements to the layout
        form_layout.addRow(self.labels[0], self.distance_edit)
        form_layout.addRow(self.labels[1], self.surge_edit)
        form_layout.addRow(QLabel('Ride Type:'), self.ride_type_combobox)

        # Create a vertical layout
        layout = QVBoxLayout()

        # Add the form layout and other elements to the layout
        layout.addLayout(form_layout)
        layout.addWidget(self.button)
        layout.addWidget(self.result_label)

        # Set the layout for the window
        self.setLayout(layout)

        # Connect the button to the predict function
        self.button.clicked.connect(self.predict_output)

    def predict_output(self):
        # Get user inputs
        input_data = [0] * len(input_vars)
        input_data[0] = float(self.distance_edit.text())
        input_data[1] = float(self.surge_edit.text())

        # Set the selected ride type to 1
        selected_ride_type = self.ride_type_combobox.currentIndex() + 2
        input_data[selected_ride_type] = 1

        # Prepare input data as a DataFrame
        input_df = pd.DataFrame([input_data], columns=input_vars)

         # Make the prediction
        prediction = xgb_model.predict(xgb.DMatrix(input_df))[0]

        # Set the result label text
        self.result_label.setText(f"The predicted output is {prediction:.2f}")

# Create the QApplication and MainWindow
app = QApplication(sys.argv)
app.setStyle('Fusion') # Set the Fusion style for the application
app.setWindowIcon(QIcon("app_icon.png"))  # Set the app icon
window = MainWindow()

# Set the window title and dimensions
window.setWindowTitle('XGBoost Model Predictor')
window.setGeometry(100, 100, 400, 600)
```

```
# Show the MainWindow
window.show()

# Run the QApplication event loop
sys.exit(app.exec_())
```

In [ ]: