Project 3

Mitch Morrison and Christian Gould

In [58]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

The Uber datset contains the following fields:

- key - a unique identifier for each trip
- fare_amount - the cost of each trip in usd
- pickup_datetime - date and time when the meter was engaged
- passenger_count - the number of passengers in the vehicle (driver entered value)
- pickup_longitude - the longitude where the meter was engaged
- pickup_latitude - the latitude where the meter was engaged
- dropoff_longitude - the longitude where the meter was disengaged
- dropoff_latitude - the latitude where the meter was disengaged

We will use the distance between pickup and dropoff, weekday, hour, and month as atttributes for prediction to the target value fare_amount.

In [244]:
```
df = pd.read_csv('../uber.csv')
df
```

Out[244]:

	Unnamed: 0	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latit
0	24238194	2015-05-07 19:52:06.0000003	7.5	2015-05-07 19:52:06 UTC	-73.999817	40.738
1	27835199	2009-07-17 20:04:56.0000002	7.7	2009-07-17 20:04:56 UTC	-73.994355	40.728
2	44984355	2009-08-24 21:45:00.00000061	12.9	2009-08-24 21:45:00 UTC	-74.005043	40.740
3	25894730	2009-06-26 08:22:21.0000001	5.3	2009-06-26 08:22:21 UTC	-73.976124	40.790
4	17610152	2014-08-28 17:47:00.000000188	16.0	2014-08-28 17:47:00 UTC	-73.925023	40.744
...	...	...	...	...	...	
199995	42598914	2012-10-28 10:49:00.00000053	3.0	2012-10-28 10:49:00 UTC	-73.987042	40.739
199996	16382965	2014-03-14 01:09:00.0000008	7.5	2014-03-14 01:09:00 UTC	-73.984722	40.736
199997	27804658	2009-06-29 00:42:00.00000078	30.9	2009-06-29 00:42:00 UTC	-73.986017	40.756
199998	20259894	2015-05-20 14:56:25.0000004	14.5	2015-05-20 14:56:25 UTC	-73.997124	40.725
199999	11951496	2010-05-15 04:08:00.00000076	14.1	2010-05-15 04:08:00 UTC	-73.984395	40.720

200000 rows × 9 columns

In [243]:
```
df = df.dropna()
df
```

Out[243]:

	Unnamed:_0	key	fare_amount	pickup_datetime	pickup_longitude	pickup_la
0	24238194	2015-05-07 19:52:06.0000003	7.5	2015-05-07 19:52:06+00:00	-73.999817	40.7
1	27835199	2009-07-17 20:04:56.0000002	7.7	2009-07-17 20:04:56+00:00	-73.994355	40.7
2	44984355	2009-08-24 21:45:00.00000061	12.9	2009-08-24 21:45:00+00:00	-74.005043	40.7
3	25894730	2009-06-26 08:22:21.0000001	5.3	2009-06-26 08:22:21+00:00	-73.976124	40.7
4	17610152	2014-08-28 17:47:00.000000188	16.0	2014-08-28 17:47:00+00:00	-73.925023	40.7
...	...	...	...	...	...	...
199995	42598914	2012-10-28 10:49:00.00000053	3.0	2012-10-28 10:49:00+00:00	-73.987042	40.7
199996	16382965	2014-03-14 01:09:00.0000008	7.5	2014-03-14 01:09:00+00:00	-73.984722	40.7
199997	27804658	2009-06-29 00:42:00.00000078	30.9	2009-06-29 00:42:00+00:00	-73.986017	40.7
199998	20259894	2015-05-20 14:56:25.0000004	14.5	2015-05-20 14:56:25+00:00	-73.997124	40.7
199999	11951496	2010-05-15 04:08:00.00000076	14.1	2010-05-15 04:08:00+00:00	-73.984395	40.7

199554 rows × 19 columns

In [62]:
```
df = df[(df.pickup_latitude<90) & (df.dropoff_latitude<90) &
        (df.pickup_latitude>-90) & (df.dropoff_latitude>-90) &
        (df.pickup_longitude<180) & (df.dropoff_longitude<180) &
        (df.pickup_longitude>-180) & (df.dropoff_longitude>-180)]
```

In [63]:
```
df.pickup_datetime=pd.to_datetime(df.pickup_datetime)

df['year'] = df.pickup_datetime.dt.year
df['month'] = df.pickup_datetime.dt.month
df['weekday'] = df.pickup_datetime.dt.day_name()
df['hour'] = df.pickup_datetime.dt.hour
```

In [64]:
```
# Ordinal Encoding days of the week to number
df['weekday_int'] = df.weekday.map({'Monday':1, 'Tuesday':2, 'Wednesday':3,
```

In [65]:
```
# One Hot encode weekday_int to isWeekend and isWeekday
df['isWeekend'] = df.weekday_int.map({1:0, 2:0, 3:0, 4:0, 5:1, 6:1, 7:1})
df['isWeekday'] = df.weekday_int.map({1:1, 2:1, 3:1, 4:1, 5:0, 6:0, 7:0})
df.head()
```

Out[65]:

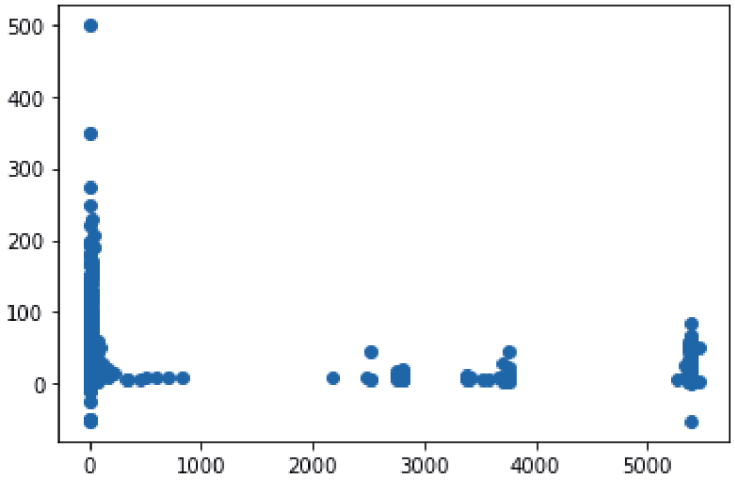
	Unnamed: 0	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude
0	24238194	2015-05-07 19:52:06.0000003	7.5	2015-05-07 19:52:06+00:00	-73.999817	40.738354
1	27835199	2009-07-17 20:04:56.0000002	7.7	2009-07-17 20:04:56+00:00	-73.994355	40.728225
2	44984355	2009-08-24 21:45:00.00000061	12.9	2009-08-24 21:45:00+00:00	-74.005043	40.740770
3	25894730	2009-06-26 08:22:21.0000001	5.3	2009-06-26 08:22:21+00:00	-73.976124	40.790844
4	17610152	2014-08-28 17:47:00.000000188	16.0	2014-08-28 17:47:00+00:00	-73.925023	40.744085

In [66]:
```
# Calculate the distance between pickup and dropoff (in feet)
from geopy import distance
df['distance']=[round(distance.distance((df.pickup_latitude[i], df.pickup_l
df.head()
```

Out[66]:

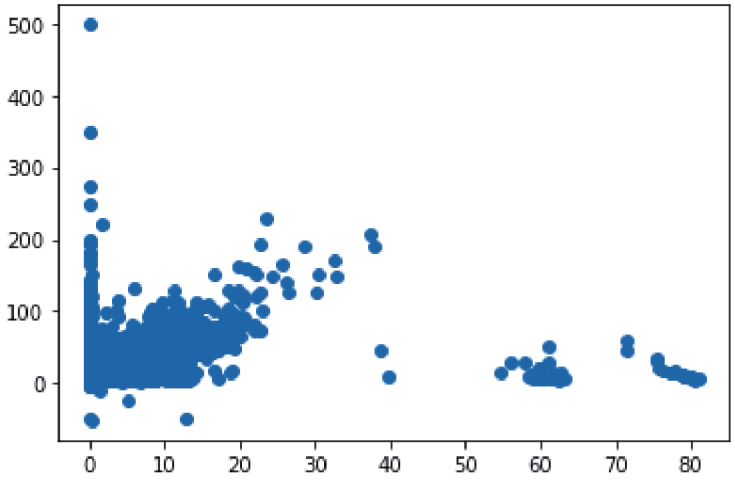
	Unnamed: 0	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude
0	24238194	2015-05-07 19:52:06.0000003	7.5	2015-05-07 19:52:06+00:00	-73.999817	40.738354
1	27835199	2009-07-17 20:04:56.0000002	7.7	2009-07-17 20:04:56+00:00	-73.994355	40.728225
2	44984355	2009-08-24 21:45:00.00000061	12.9	2009-08-24 21:45:00+00:00	-74.005043	40.740770
3	25894730	2009-06-26 08:22:21.0000001	5.3	2009-06-26 08:22:21+00:00	-73.976124	40.790844
4	17610152	2014-08-28 17:47:00.000000188	16.0	2014-08-28 17:47:00+00:00	-73.925023	40.744085

In [67]:
```
# plot distance vs fare of rides to see outliers
plt.scatter(df['distance'], df['fare_amount'])
plt.show()
```



In [68]:
```
# drop all rows with distance > 100 miles
df = df[df.distance<100]
```

In [77]:
```
# plot distance vs fare of rides again
plt.scatter(df['distance'], df['fare_amount'])
plt.show()
```



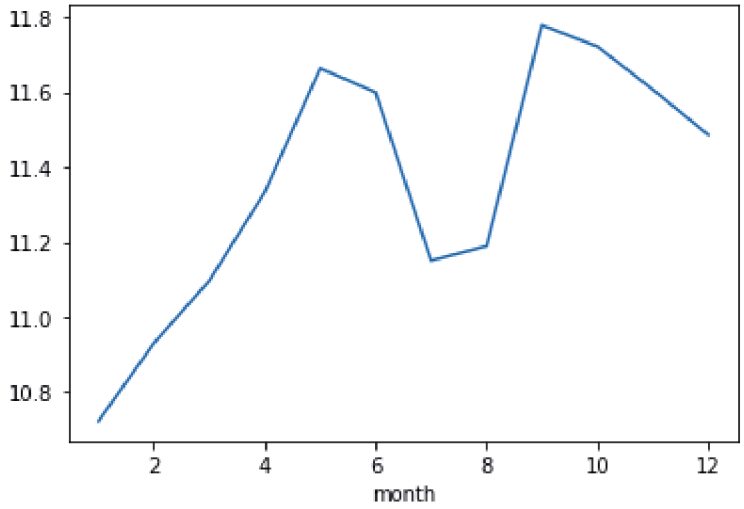
In [159]:
```
# normalize the distance field using min max scaling
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
df['distance_norm'] = scaler.fit_transform(df[['distance']])
df
```

Out[159]:

	Unnamed: 0	key	fare_amount	pickup_datetime	pickup_longitude	pickup_la
0	24238194	2015-05-07 19:52:06.0000003	7.5	2015-05-07 19:52:06+00:00	-73.999817	40.7
1	27835199	2009-07-17 20:04:56.0000002	7.7	2009-07-17 20:04:56+00:00	-73.994355	40.7
2	44984355	2009-08-24 21:45:00.00000061	12.9	2009-08-24 21:45:00+00:00	-74.005043	40.7
3	25894730	2009-06-26 08:22:21.0000001	5.3	2009-06-26 08:22:21+00:00	-73.976124	40.7
4	17610152	2014-08-28 17:47:00.000000188	16.0	2014-08-28 17:47:00+00:00	-73.925023	40.7
...	...	...	...	...	...	...
199995	42598914	2012-10-28 10:49:00.00000053	3.0	2012-10-28 10:49:00+00:00	-73.987042	40.7
199996	16382965	2014-03-14 01:09:00.0000008	7.5	2014-03-14 01:09:00+00:00	-73.984722	40.7
199997	27804658	2009-06-29 00:42:00.00000078	30.9	2009-06-29 00:42:00+00:00	-73.986017	40.7
199998	20259894	2015-05-20 14:56:25.0000004	14.5	2015-05-20 14:56:25+00:00	-73.997124	40.7
199999	11951496	2010-05-15 04:08:00.00000076	14.1	2010-05-15 04:08:00+00:00	-73.984395	40.7

199554 rows × 19 columns

In [197]:
```
# plot fare amount by month
df.groupby('month').fare_amount.mean().plot()
plt.show()
```



Since we can see that the month has an influence on the price we will include this feature in our models.

In [209]:
```
# create a custom transformer
from sklearn.model_selection import train_test_split
# split the df into train and test

cols = []
for i in df.columns.values:
    cols.append(i.replace(' ','_'))

df.columns = cols
target = "fare_amount"

X = df.drop([target],axis=1)
Y = df[target]

Train_X, Test_X, Train_Y, Test_Y = train_test_split(X, Y, train_size=0.8, t
Train_X.reset_index(drop=True,inplace=True)
```

Creating our models and using them for prediction

In [210]:
```
# import error metrics
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
```

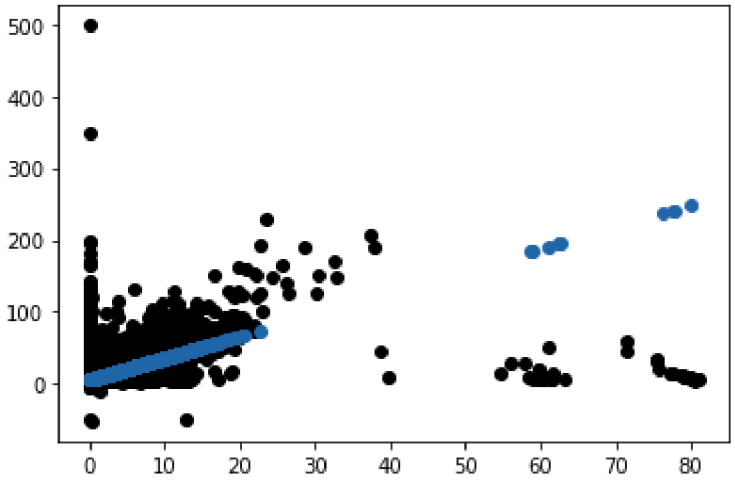
In [223]:
```
# run Linear regression on the train data to predict fare_amount
from sklearn.linear_model import LinearRegression
```

In [224]:
```
lin = LinearRegression().fit(Train_X['distance'].values.reshape(-1,1), Trai
preds = lin.predict(Test_X['distance'].values.reshape(-1,1))
print("Mean squared error: %.2f" % mean_squared_error(Test_Y, preds))
print("Mean absolute error: %.2f" % mean_absolute_error(Test_Y, preds))
```

```
Mean squared error: 42.65
Mean absolute error: 2.85
```

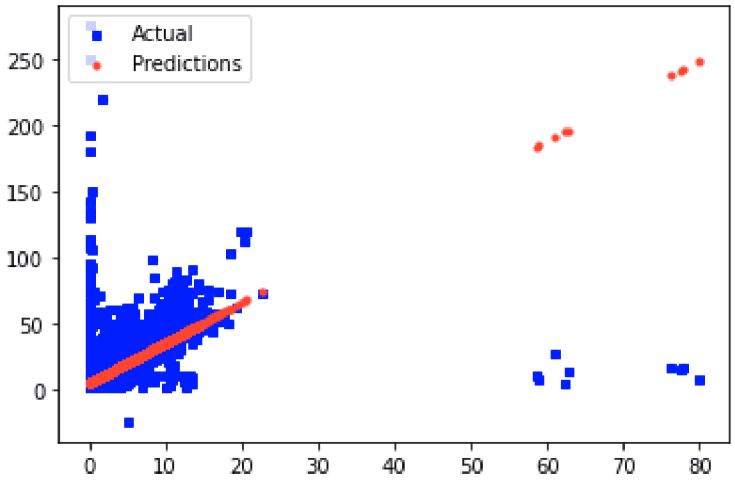
In [212]:
```
def plot_it(x, y1, y2):# plot predictions vs actual
    fig = plt.figure()
    ax1 = fig.add_subplot(111)

    ax1.scatter(x, y1, s=10, c='b', marker="s", label='Actual')
    ax1.scatter(x, y2, s=10, c='r', marker="o", label='Predictions')
    plt.legend(loc='upper left');
    plt.show()
```

In [213]:
```
# plot the train data and the predicted values
plt.scatter(Train_X['distance'], Train_Y,  color='black')
plt.scatter(Test_X['distance'], preds)
plt.show()
```



In [214]:
```
# try prediction using distance_norm and all x features instead of distance
norm = LinearRegression().fit(Train_X[['distance', 'isWeekend', 'hour', 'mo
norm_preds = norm.predict(Test_X[['distance', 'isWeekend', 'hour', 'month']
print("Mean squared error: %.2f" % mean_squared_error(Test_Y, norm_preds))
print("Mean absolute error: %.2f" % mean_absolute_error(Test_Y, norm_preds))
plot_it(Test_X['distance'], Test_Y, preds)
```

```
Mean squared error: 42.59
Mean absolute error: 2.84
```



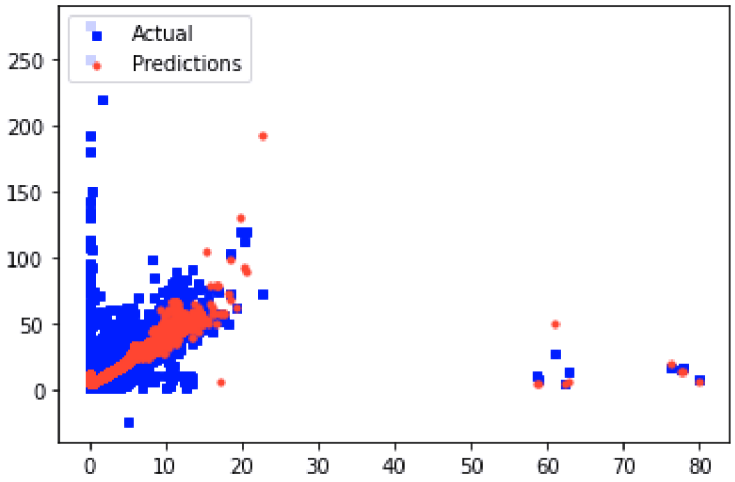
Although the linear regression does not perform poorly on the sparse data, it does not very well capture any nuances in the data. It performs very poorly on long rides data and can severely underestimate some short rides at high costs, ultimately creating a significant error value.

We can see that given the MSE using each distance and distance_norm as the independant variable has no impact on the performance of the linear regression. We will continue to use the non-normalized values because the actual cost amounts will be easier to visualize and comprehend from a person's POV.

In [215]:
```
from sklearn.tree import DecisionTreeRegressor
```

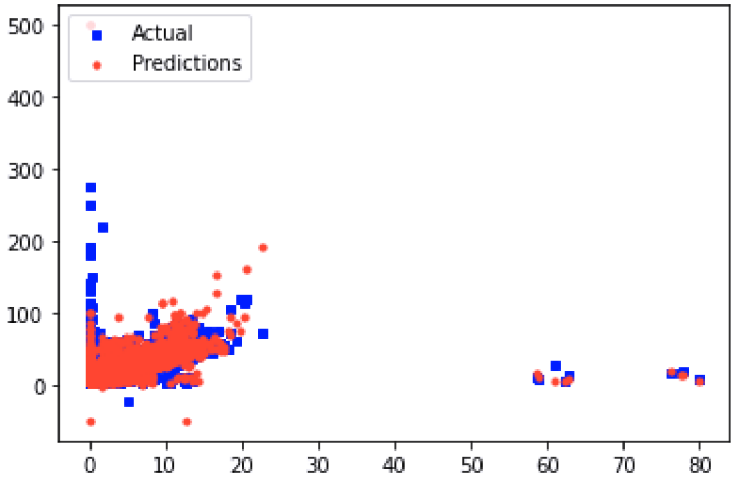
In [216]:
```
# using only distance data for X
regressor = DecisionTreeRegressor(random_state=0)
regressor.fit(Train_X['distance'].values.reshape(-1,1), Train_Y)
preds = regressor.predict(Test_X['distance'].values.reshape(-1,1))
print("Mean squared error: %.2f" % mean_squared_error(Test_Y, preds))
print("Mean absolute error: %.2f" % mean_absolute_error(Test_Y, preds))
plot_it(Test_X['distance'], Test_Y, preds)
```

```
Mean squared error: 29.74
Mean absolute error: 2.49
```



In [217]:
```
# using multiple x attributes
regressor = DecisionTreeRegressor(random_state=0)
regressor.fit(Train_X[['distance', 'isWeekend', 'hour', 'month']], Train_Y)
preds = regressor.predict(Test_X[['distance', 'isWeekend', 'hour', 'month']
print("Mean squared error: %.2f" % mean_squared_error(Test_Y, preds))
print("Mean absolute error: %.2f" % mean_absolute_error(Test_Y, preds))
plot_it(Test_X['distance'], Test_Y, preds)
```

```
Mean squared error: 58.50
Mean absolute error: 3.31
```



It seems that the decision tree regressor performs much better on the outliar data, perhaps a result of overfitting. Also notable though, is that long rides (alhough some may just be bad data) are all quite low cost indicating there may be some kind of minimum price for distance or some deals they

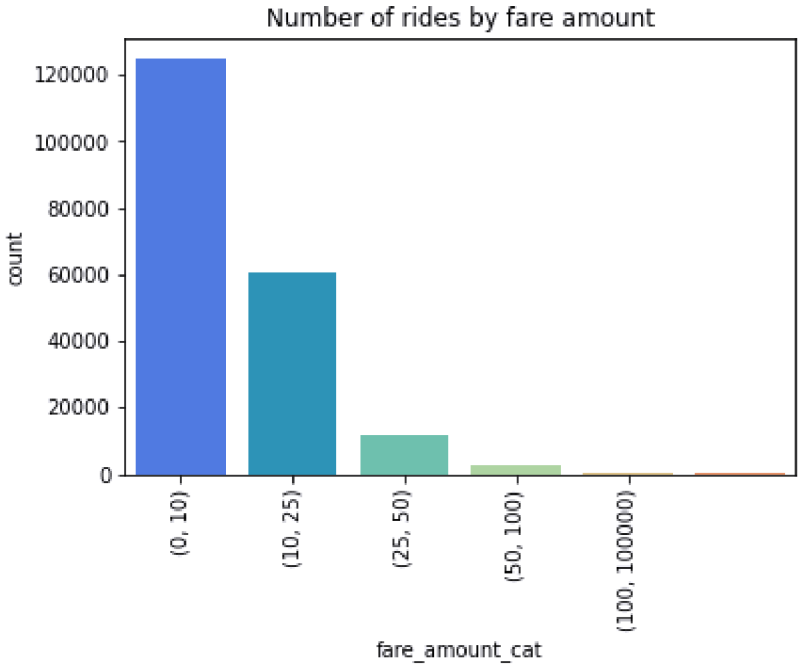
may have had.

Comparing using a single vs multiple data attributes our error increases but we can also see that our predictions on the graph span a larger area which is more comprehensive of our data. Using only distance data, the results are near linear because it has little features to work with.

In [218]:
```
se the random forest classifier, our y-data needs to not be continuous
ill cut the y data into 5 categories of price ranges (0-10, 10-25, 25-50, 5
 df to not affect the original df
y = df.copy()
nges = [(0, 10), (10, 25), (25, 50), (50, 100), (100, 100000)]
 pd.IntervalIndex.from_tuples(bin_ranges)
bins)
y['fare_amount_cat'] = pd.cut(df_copy['fare_amount'], bins)

klearn import preprocessing
encoder = preprocessing.LabelEncoder()
y['fare_amount_cat']= label_encoder.fit_transform(df_copy['fare_amount_cat'
y.head()


t data into train and test
X_fr, Test_X_fr, Train_Y_fr, Test_Y_fr = train_test_split(df_copy, df_copy[
```

```
IntervalIndex([(0, 10], (10, 25], (25, 50], (50, 100], (100, 100000]], dt
ype='interval[int64, right]')
```

In [219]:
```
import seaborn as sns
sns.countplot(x='fare_amount_cat',data=df_copy, palette='rainbow')
plt.xticks(ticks=range(0, 5), labels=bin_ranges, rotation=90)
plt.title("Number of rides by fare amount")
plt.show()
```



Now that we see the distribution of fares across the different cut categories we have created, we

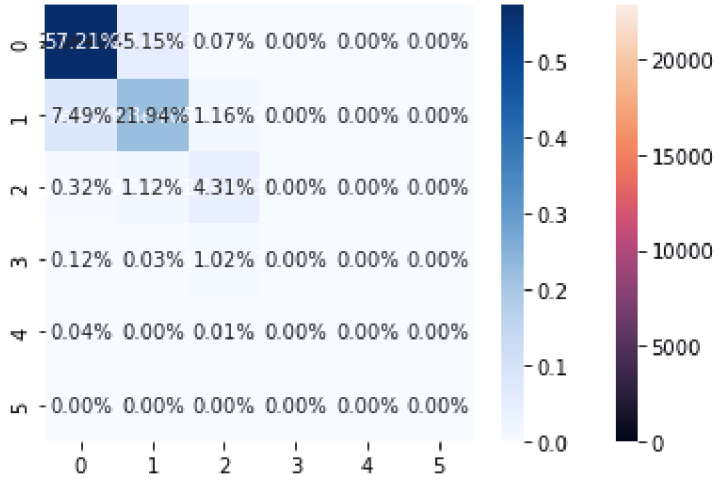
can try and predict the fare_amount groups using our distance data.

In [220]:
```
from sklearn.ensemble import RandomForestClassifier

regressor = RandomForestClassifier(max_depth=2, random_state=0)
regressor.fit(Train_X_fr['distance'].values.reshape(-1, 1), Train_Y_fr)
preds_fr = regressor.predict(Test_X_fr['distance'].values.reshape(-1, 1))
print("Mean squared error: %.2f" % mean_squared_error(Test_Y_fr, preds_fr))
print("Mean absolute error: %.2f" % mean_absolute_error(Test_Y_fr, preds_fr
```

```
Mean squared error: 0.19
Mean absolute error: 0.17
```

In [221]:
```
# import confusion matrix
from sklearn.metrics import confusion_matrix
cf_matrix = confusion_matrix(Test_Y_fr, preds_fr)
sns.heatmap(cf_matrix, annot=True)
sns.heatmap(cf_matrix/np.sum(cf_matrix), annot=True,
            fmt='.2%', cmap='Blues')
```

Out[221]: <AxesSubplot:>

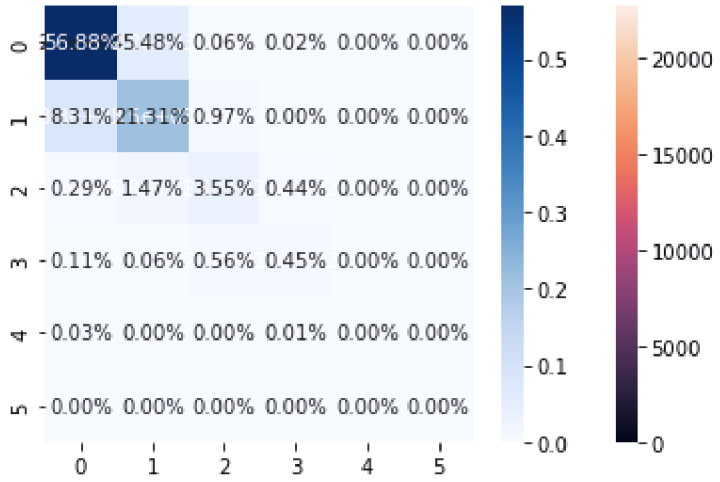


The heatmap visualization above tells us that our decision tree is predicting 0 correctly 57% of the time, and an additional 7% of the time it predicts 0 when it is infact a 1. Similarly, the regression predicts 1 correctly 22% of the time of overall predictions and a value of 0 incorrectly on a total of 5% of all total predictions. It seems that this classifier has a solid guess of the price of the ride but is in the incorrect range 7 + 5 + 1 + 1 + 1 + ... = 15% of te time. The majority of the predictions it makes are in the accurate range.

In [222]:
```
# KNN Classifier
from sklearn.neighbors import KNeighborsClassifier

model = KNeighborsClassifier(n_neighbors=5)

# Train the model using the training sets
model.fit(Train_X_fr[['distance', 'isWeekend', 'hour', 'month']], Train_Y_f

#Predict Output
predicted = model.predict(Test_X_fr[['distance', 'isWeekend', 'hour', 'mont
print("Mean squared error: %.2f" % mean_squared_error(Test_Y_fr, predicted)
print("Mean absolute error: %.2f" % mean_absolute_error(Test_Y_fr, predicte

cf_matrix = confusion_matrix(Test_Y_fr, predicted)
sns.heatmap(cf_matrix, annot=True)
sns.heatmap(cf_matrix/np.sum(cf_matrix), annot=True,
            fmt='.2%', cmap='Blues')
```

```
Mean squared error: 0.21
Mean absolute error: 0.19
```

Out[222]: <AxesSubplot:>



In [ ]:

Now that we have created and used our models for our dataset we can begin to compare their performances. Firstly, the models using the continuous data cannot be compared accurately to the models using discrete interval data. This is only fair because they are supplied much different data.

The first two models are **Linear Regression and Decision Tree regression**. AS expected the linear regression uses a straight forward MSE error approach to creating a line of best fit. It returns a MSE value of ~42 and MAE of < 3 for both the approach using just distance as the independant variable as well as the appraoch using distance, hour, isWeekday, and month.

The decision tree regressions performed very differently with multiple features - the model with just distance had a near linear approach to the majority of the data and overfit the outliars, reducing its MSE to ~20 and MAE to 2.5. The model with all four feature columns has a MSE of 58 and MAE of 3. The additional features made our model expand its predictions, particularly in the first 20 miles of driving. The graph detailing its predictions shows a clear use of all the features as it tries to better predict fares for shorter rides, whereas a linear approach sacrifices closeness on individual instances for a lower error overall.

The next two models we will be looking at are the **Random Forest Classifier and K Neighbors classifier**. Each of these models predicts categorical data, not continous values. Given this requirement, I had to adjust our target data (fare amount) into categeories (using pd.cut) based on the price range. I made 5 groups to indicate short (0, 10], medium short (10, 25], medium (25, 50], medium long (50, 100], and long (100, 100000]rides based on their PRICE RANGE (not distance). This allowed our classifiers to take in our distance, hour, month, and isWeekend data to predict the categories for price range.

The random forest classifier had a MSE of .19 meaning that the average error between the category that was and should be predicted is that value. The MAE is at .17. This is just slightly better than the KNN classifier which comes in at a MSE of .21 and MAE of .19. Adjusting the number of neighbors on the KNN classifier can both improve and worsent he MSE and MAE values.

The best way to see the difference between the two prediction sets is to look at the confusion matrices printed below each of their respective cells. From the visualizations we can see that most commonly the classifiers predict either 1 or 2 when it is the opposite. There are almost no predictions to the 4 and 5 class since the training data lacks much representation from these price ranges.

Using Grid search to create a calibrated and tuned model

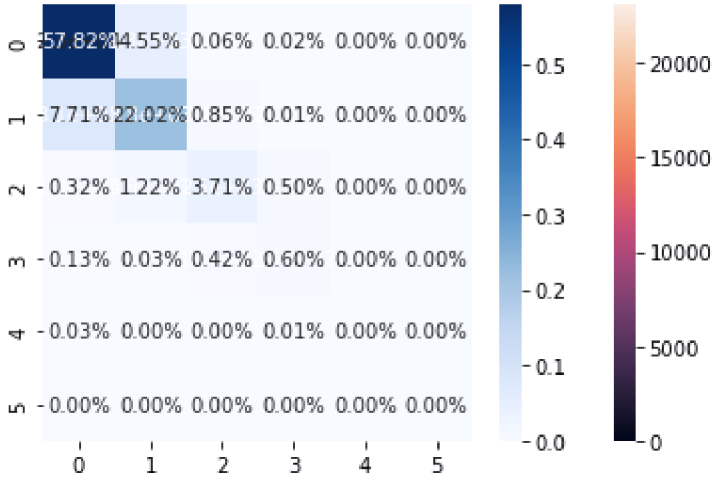
In [225]:
```
# the GridSearchCV allows us to find the best parameters for the model
# we will use the same parameters as before
# CalibratedClassifierCV is a wrapper around the calibration algorithms

from sklearn.model_selection import GridSearchCV
from sklearn.calibration import CalibratedClassifierCV

calibrated_forest = CalibratedClassifierCV(
   base_estimator=RandomForestClassifier(n_estimators=10))
param_grid = {
   'base_estimator__max_depth': [2, 4, 6, 8]}
search = GridSearchCV(calibrated_forest, param_grid, cv=5)
search.fit(Train_X_fr[['distance', 'isWeekend', 'hour', 'month']], Train_Y_
```

Out[225]:
```
GridSearchCV(cv=5,
             estimator=CalibratedClassifierCV(base_estimator=RandomForest
Classifier(n_estimators=10)),
             param_grid={'base_estimator__max_depth': [2, 4, 6, 8]})
```

In [228]:
```
results = search.predict(Test_X_fr[['distance', 'isWeekend', 'hour', 'month
print("Mean squared error: %.2f" % mean_squared_error(Test_Y_fr, results))
print("Mean absolute error: %.2f" % mean_absolute_error(Test_Y_fr, results)

cf_matrix = confusion_matrix(Test_Y_fr, results)
sns.heatmap(cf_matrix, annot=True)
sns.heatmap(cf_matrix/np.sum(cf_matrix), annot=True,
            fmt='.2%', cmap='Blues')
```

```
Mean squared error: 0.19
Mean absolute error: 0.17
```

Out[228]:
```
<AxesSubplot:>
```



Given the results of our model we can see that our performance improved slightly (by looking at the Confusion Matrix). Although the MSE and MAE did not change, we can see that the total correct predictions percentages for both class 1 and 2 are higher than before.

The grid search updated the base estimators feature used in our hyperparameter grid.

Wrap up

Overall in this program we executed from start to finish the read in, EDA, modeling, and performance evaluation (and hyperparameter tuning) of the Uber fares dataset. These steps required us to understand our data and how each model performs, such as how it can take single or multiple attributes of x data or if the target variable needs to be categorical or can be continuous. In this particular notebook, the following items from the checklist have been completed.

In [255]:
```
# ! pip install openpyxl
completed = pd.read_excel('../P3Spreadsheet.xlsx')
completed = completed.dropna(subset=["Task"])
completed
```

Out[255]:

	Task	Uber Fares Dataset
0	Clean the Data (deal with missing values)	x
1	Use an Ordinal Encoder	x
2	Use a One Hot Encoder	NaN
3	Implement a custom transformer	x
4	Scale/normalize/standardize features using skl...	x
6	Use SGDClassifier	NaN
7	Use sklearn.linear_model.LinearRegression	x
8	Use sklearn.tree.DecisionTreeRegressor	x
9	Use sklearn.ensamble.RandomForestClassifier	x
10	Use sklearn.neighbors.KNeighborsClassifier	x
11	use OvO or OvR classifier	NaN
13	Use k-fold Cross Validation (cross_val_score)	NaN
14	Use StratifiedKFold cross validation	NaN
16	Use sklearn.metrics.mean_squared_error and at ...	x
17	Generate a confusion matrix	x
18	Generate a ROC Curve or related	NaN
19	Use Grid Search CV or RandomizedSearch CV to t...	x
21	Use an Ensamble of Methods	x
23	Evaluate your system on the Test Data	x
25	Create a single pipeline that does full proces...	x