# House Price Prediction

**X1=** the transaction date (for example, 2013.250=2013 March, 2013.500=2013 June, etc.)
**X2=** the house age (unit: year)
**X3=** the distance to the nearest MRT station (unit: meter)
**X4=** the number of convenience stores in the living circle on foot (integer)
**X5=** the geographic coordinate, latitude. (unit: degree)
**X6=** the geographic coordinate, longitude. (unit: degree)
**Y=** house price of unit area (10000 New Taiwan Dollar/Ping, where Ping is a local unit, 1 Ping = 3.3 meter squared)

## Data Pre-processing

#### Load data

In [1]:

```
# Import relevant packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:

```
# Loading the data
housing = pd.read_excel('real_estate_valuation.xlsx')

# Converting to dataframe
df_housing = pd.DataFrame(housing)

# Making a copy of the original dataframe to work on
df = housing.copy(deep=True)
```

In [3]:

```
# previewing the firs 5 rows
df.head()
```

Out[3]:

| | No | X1 transaction date | X2 house age | X3 distance to the nearest MRT station | X4 number of convenience stores | X5 latitude | X6 longitude | Y house price of unit area |
|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 2012.916667 | 32.0 | 84.87882 | 10 | 24.98298 | 121.54024 | 37.9 |
| **1** | 2 | 2012.916667 | 19.5 | 306.59470 | 9 | 24.98034 | 121.53951 | 42.2 |
| **2** | 3 | 2013.583333 | 13.3 | 561.98450 | 5 | 24.98746 | 121.54391 | 47.3 |
| **3** | 4 | 2013.500000 | 13.3 | 561.98450 | 5 | 24.98746 | 121.54391 | 54.8 |
| **4** | 5 | 2012.833333 | 5.0 | 390.56840 | 5 | 24.97937 | 121.54245 | 43.1 |

In [4]:

```
# Previewing the last 5 rows
df.tail()
```

Out[4]:

| | No | X1 transaction date | X2 house age | X3 distance to the nearest MRT station | X4 number of convenience stores | X5 latitude | X6 longitude | Y house price of unit area |
|---|---|---|---|---|---|---|---|---|
| **409** | 410 | 2013.000000 | 13.7 | 4082.01500 | 0 | 24.94155 | 121.50381 | 15.4 |
| **410** | 411 | 2012.666667 | 5.6 | 90.45606 | 9 | 24.97433 | 121.54310 | 50.0 |
| **411** | 412 | 2013.250000 | 18.8 | 390.96960 | 7 | 24.97923 | 121.53986 | 40.6 |
| **412** | 413 | 2013.000000 | 8.1 | 104.81010 | 5 | 24.96674 | 121.54067 | 52.5 |
| **413** | 414 | 2013.500000 | 6.5 | 90.45606 | 9 | 24.97433 | 121.54310 | 63.9 |

In [5]:

```
# preview the number of attributes
df.shape[1]
```

Out[5]:

8

In [6]:

```
# Preview the column names
df.columns
```

Out[6]:

```
Index(['No', 'X1 transaction date', 'X2 house age',
       'X3 distance to the nearest MRT station',
       'X4 number of convenience stores', 'X5 latitude', 'X6 longitud
e',
       'Y house price of unit area'],
      dtype='object')
```

In [7]:

```
# Checking for missing values
df.isna().sum()
```

Out[7]:

```
No                                        0
X1 transaction date                       0
X2 house age                              0
X3 distance to the nearest MRT station    0
X4 number of convenience stores           0
X5 latitude                               0
X6 longitude                              0
Y house price of unit area                0
dtype: int64
```

In [8]:

```
# No missing values within this dataset, no imputer required.
```

In [9]:

```
# checking the datatypes of each column
df.dtypes
```

Out[9]:

```
No                                          int64
X1 transaction date                       float64
X2 house age                              float64
X3 distance to the nearest MRT station    float64
X4 number of convenience stores             int64
X5 latitude                               float64
X6 longitude                              float64
Y house price of unit area                float64
dtype: object
```

In [10]:

```
# Summarising the dataframe
df.describe()
```

Out[10]:

| | No | X1 transaction date | X2 house age | X3 distance to the nearest MRT station | X4 number of convenience stores | X5 latitude | X6 longitude |
|---|---|---|---|---|---|---|---|
| **count** | 414.000000 | 414.000000 | 414.000000 | 414.000000 | 414.000000 | 414.000000 | 414.000000 |
| **mean** | 207.500000 | 2013.148953 | 17.712560 | 1083.885689 | 4.094203 | 24.969030 | 121.533361 |
| **std** | 119.655756 | 0.281995 | 11.392485 | 1262.109595 | 2.945562 | 0.012410 | 0.015347 |
| **min** | 1.000000 | 2012.666667 | 0.000000 | 23.382840 | 0.000000 | 24.932070 | 121.473530 |
| **25%** | 104.250000 | 2012.916667 | 9.025000 | 289.324800 | 1.000000 | 24.963000 | 121.528085 |
| **50%** | 207.500000 | 2013.166667 | 16.100000 | 492.231300 | 4.000000 | 24.971100 | 121.538630 |
| **75%** | 310.750000 | 2013.416667 | 28.150000 | 1454.279000 | 6.000000 | 24.977455 | 121.543305 |
| **max** | 414.000000 | 2013.583333 | 43.800000 | 6488.021000 | 10.000000 | 25.014590 | 121.566270 |

### Cleaning the data

In [11]:

```
# removing irrelvant column
df.drop(columns = 'No', inplace = True)
```

In [12]:

```
# Cleaning the column names
df.rename(columns = {'X1 transaction date': 'transaction_date',
                     'X2 house age': 'house_age',
                     'X3 distance to the nearest MRT station':'mrt_distance',
                     'X4 number of convenience stores':'no_of_stores',
                     'X5 latitude': 'latitude',
                     'X6 longitude': 'longitude',
                     'Y house price of unit area':'house_price'},inplace = True)
```

In [13]:

```
df.head()
```

Out[13]:

| | transaction_date | house_age | mrt_distance | no_of_stores | latitude | longitude | house_price |
|---|---|---|---|---|---|---|---|
| **0** | 2012.916667 | 32.0 | 84.87882 | 10 | 24.98298 | 121.54024 | 37.9 |
| **1** | 2012.916667 | 19.5 | 306.59470 | 9 | 24.98034 | 121.53951 | 42.2 |
| **2** | 2013.583333 | 13.3 | 561.98450 | 5 | 24.98746 | 121.54391 | 47.3 |
| **3** | 2013.500000 | 13.3 | 561.98450 | 5 | 24.98746 | 121.54391 | 54.8 |
| **4** | 2012.833333 | 5.0 | 390.56840 | 5 | 24.97937 | 121.54245 | 43.1 |

### Preliminary Analysis

In [14]:

```
# What's the lowest house price
print(f"The lowest house price is {round(df.house_price.min(),2)}/m\u00b2")
```

```
The lowest house price is 7.6/m²
```

In [15]:

```
print(f"The highest house price is {round(df.house_price.max(),2)}/m\u00b2")
```

```
The highest house price is 117.5/m²
```

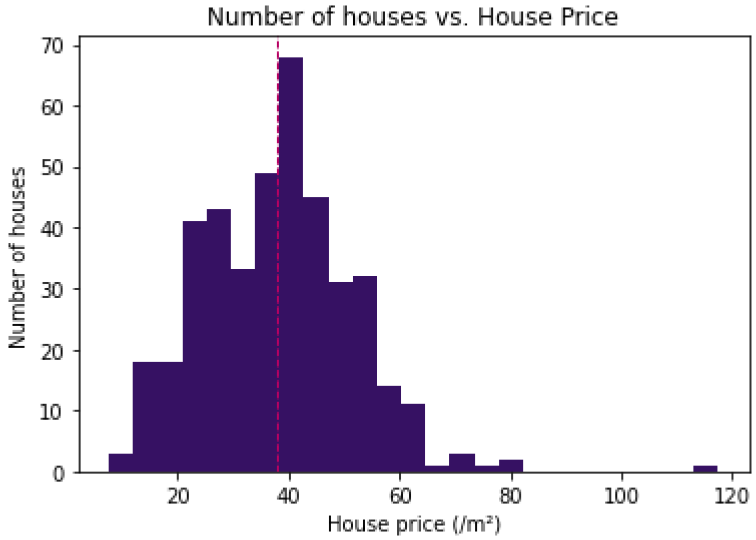
In [16]:

```
print(f"The average house price is {round(df.house_price.mean(),2)}/m\u00b2")
```

```
The average house price is 37.98/m²
```

In [17]:

```
# Checking the distribution of the target variable (house_price)
plt.hist(df["house_price"], bins=25, color="#361163")
plt.xlabel("House price (/m\u00b2)")
plt.ylabel("Number of houses")
plt.title("Number of houses vs. House Price")
plt.ticklabel_format(style='plain')

# Plotting the line for average house price (house_price)
plt.axvline(df["house_price"].mean(), color='#B70062', linestyle='dashed', linewidth
plt.show()

# ===== ADD COMMA TO VALUES
# ===== EXPAND THE TICKS?
```




This plot shows the house price (house_price) and the number of houses sold in the price range. It is skewed right, where the right side is the highest house price sold, with only a one houses selling for 117.5/m². Most houses sold between 10/m² - 65/m². Around 15 houses' sale price was between 60/m² - 80/m².

In [18]:

```
# What's the average distance to the nearest MRT station in meters?
print("The average distance to the nearest MRT station is",
      round(df.mrt_distance.mean(),2),
      "meters")
```

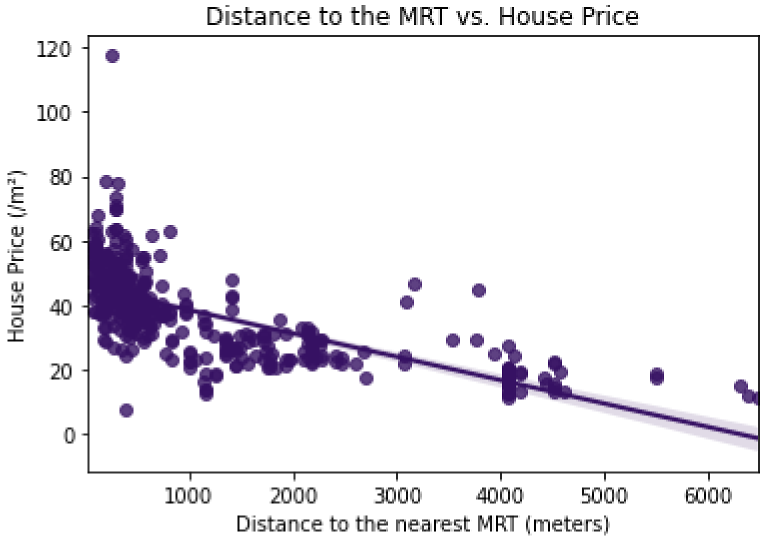
```
The average distance to the nearest MRT station is 1083.89 meters
```

In [19]:

```
# Does the distance to the nearest MRT have an impact on the house price?
sns.regplot(x="mrt_distance", y="house_price", data=df, color="#361163")
plt.ticklabel_format(style='plain')
plt.title('Distance to the MRT vs. House Price')
plt.xlabel('Distance to the nearest MRT (meters)')
plt.ylabel('House Price (/m\u00b2)')

# ====== HIGHLIGHT THE OUTLIERS
```

Out[19]:

```
Text(0, 0.5, 'House Price (/m²)')
```




In [20]:

```
# Investigating outlier 1
df[df['house_price'] == df.house_price.max()]
```

Out[20]:

| | transaction_date | house_age | mrt_distance | no_of_stores | latitude | longitude | house_price |
|---|---|---|---|---|---|---|---|
| **270** | 2013.333333 | 10.8 | 252.5822 | 1 | 24.9746 | 121.53046 | 117.5 |

In [21]:

```
# Investigating outlier 2
df[df['house_price'] == df.house_price.min()]
```

Out[21]:

| | transaction_date | house_age | mrt_distance | no_of_stores | latitude | longitude | house_price |
|---|---|---|---|---|---|---|---|
| **113** | 2013.333333 | 14.8 | 393.2606 | 6 | 24.96172 | 121.53812 | 7.6 |

- It seems that the houses with the shortest distance to the MRT station have a higher house price.
- It is also evident that the houses with a shorter distance to the MRT station, sold more than those that are further away from the nearest MRT station.
- The outlier on the plot shows the house with highest price overall sold at 117.5/m², situated 252.58 meters away to the nearest MRT station.
- However, another outlier shows that the house sold at the lowest price was situated close to an MRT station with only 393.26 meters away.
- Nevertheless, the distance to the nearest MRT station (mrt_distance) is likely to be a strong predictor for the house price (house_price)

In [22]:

```
# Removing outliers to make to prevent skewed results
df.drop(labels=270, axis=0, inplace = True)
df.drop(labels=113, axis=0, inplace = True)
```

In [23]:

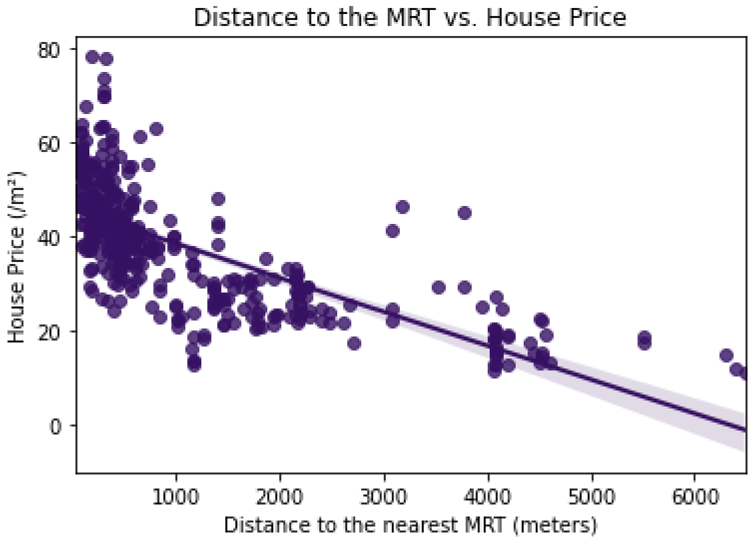
```
# Checking the outliers have been dropped
df.shape[0]
```

Out[23]:

```
412
```

In [24]:

```
# Rechecking the previous plot
sns.regplot(x="mrt_distance", y= "house_price", data=df, color="#361163")
plt.title('Distance to the MRT vs. House Price')
plt.xlabel('Distance to the nearest MRT (meters)')
plt.ylabel('House Price (/m\u00b2)')
plt.show()
```

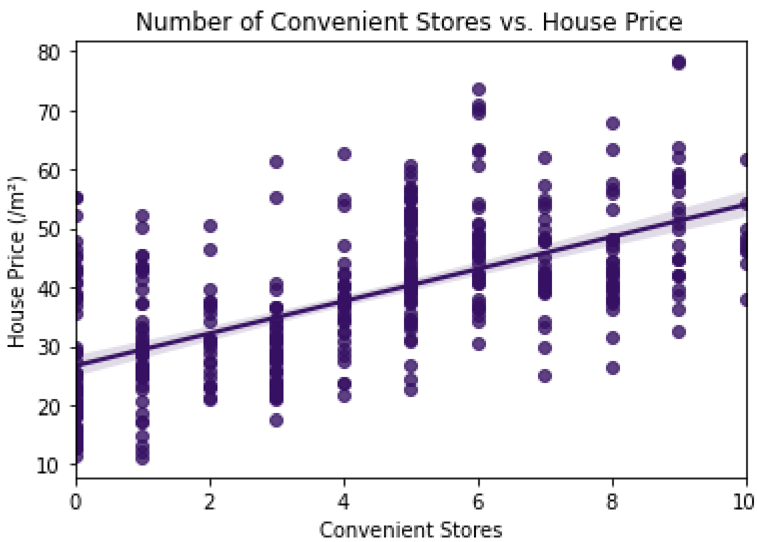



In [25]:

```
# Does having more convenient stores near the living circle increase the house price
sns.regplot(x="no_of_stores", y="house_price", data=df, color="#361163")
plt.title('Number of Convenient Stores vs. House Price')
plt.xlabel('Convenient Stores')
plt.ylabel('House Price (/m\u00b2)')
```

Out[25]:

```
Text(0, 0.5, 'House Price (/m²)')
```



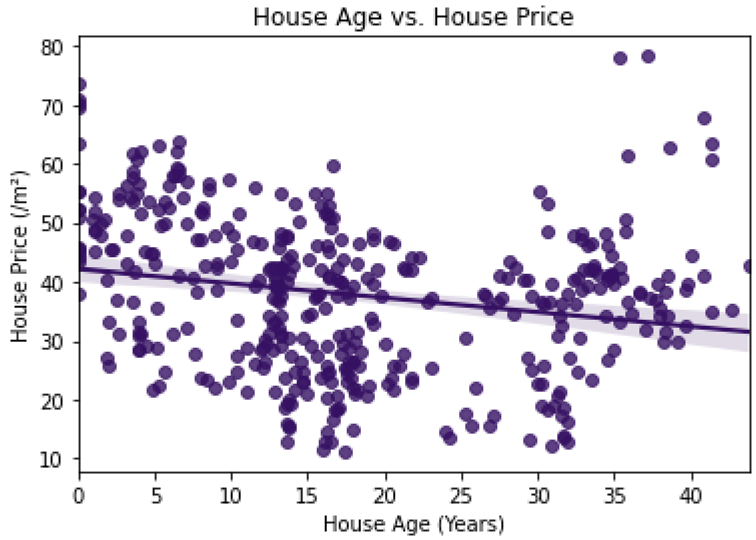

- This plot shows the number of the convenient stores in the living circle (no_of_stores) has a positive linear trend and could be a good predictor for the house price (house_price)

In [26]:

```
# Does the age of the house have impact on the sale price?
sns.regplot(x="house_age", y="house_price", data=df, color="#361163")
plt.title('House Age vs. House Price')
plt.xlabel('House Age (Years)')
plt.ylabel('House Price (/m\u00b2)')
```

Out[26]:

```
Text(0, 0.5, 'House Price (/m²)')
```

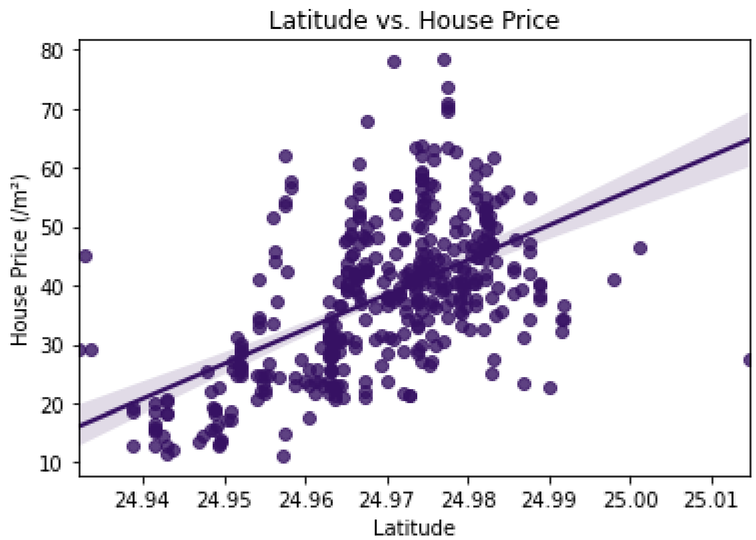



In [27]:

```
# Does latitude have impact on the sale price?
sns.regplot(x="latitude", y="house_price", data=df, color="#361163")
plt.title('Latitude vs. House Price')
plt.xlabel('Latitude')
plt.ylabel('House Price (/m\u00b2)')
```

Out[27]:

```
Text(0, 0.5, 'House Price (/m²)')
```

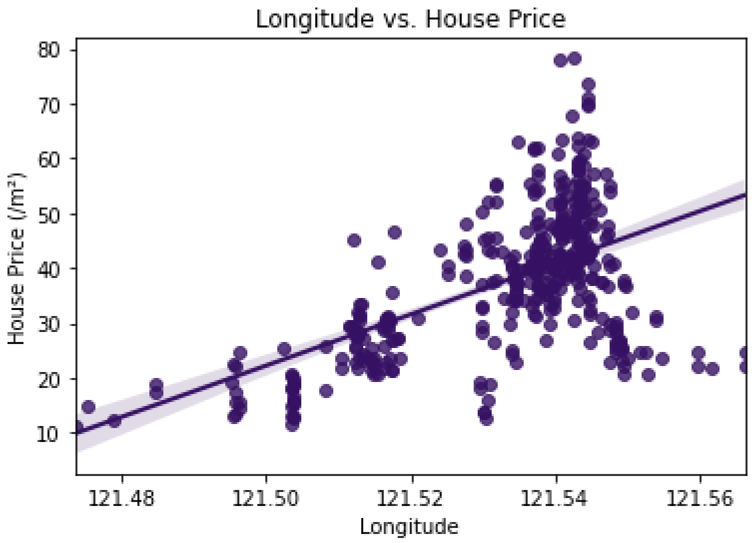


In [28]:

```
# Does longitude have impact on the sale price?
sns.regplot(x="longitude", y="house_price", data=df, color="#361163")
plt.title('Longitude vs. House Price')
plt.xlabel('Longitude')
plt.ylabel('House Price (/m\u00b2)')
```

Out[28]:

```
Text(0, 0.5, 'House Price (/m²)')
```

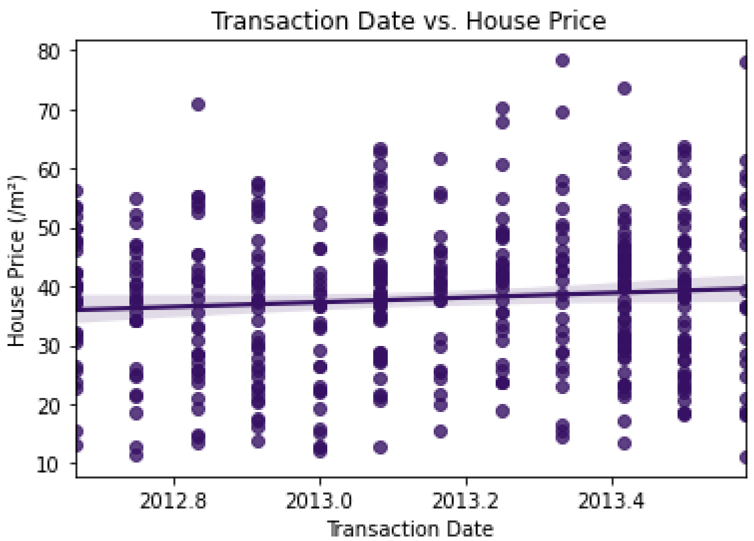



In [29]:

```
# How does the transaction date have impact on the sale price?
sns.regplot(x="transaction_date", y="house_price", data=df, color="#361163")
plt.title('Transaction Date vs. House Price')
plt.xlabel('Transaction Date')
plt.ylabel('House Price (/m\u00b2)')
```

Out[29]:

```
Text(0, 0.5, 'House Price (/m²)')
```



In [30]:

```
df.head()
```

Out[30]:

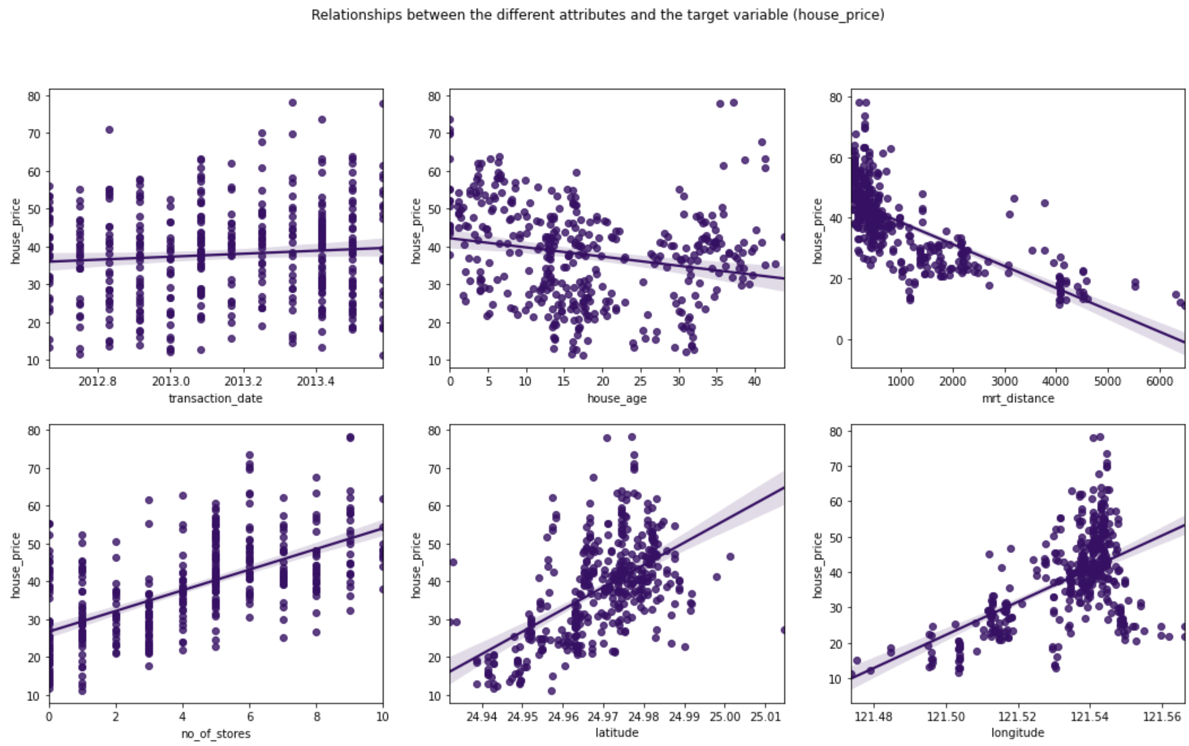
| | transaction_date | house_age | mrt_distance | no_of_stores | latitude | longitude | house_price |
|---|---|---|---|---|---|---|---|
| **0** | 2012.916667 | 32.0 | 84.87882 | 10 | 24.98298 | 121.54024 | 37.9 |
| **1** | 2012.916667 | 19.5 | 306.59470 | 9 | 24.98034 | 121.53951 | 42.2 |
| **2** | 2013.583333 | 13.3 | 561.98450 | 5 | 24.98746 | 121.54391 | 47.3 |
| **3** | 2013.500000 | 13.3 | 561.98450 | 5 | 24.98746 | 121.54391 | 54.8 |
| **4** | 2012.833333 | 5.0 | 390.56840 | 5 | 24.97937 | 121.54245 | 43.1 |

In [31]:

```
fig, axes = plt.subplots(2, 3, figsize=(18, 10))

fig.suptitle('Relationships between the different attributes and the target variable

sns.regplot(ax=axes[0, 0], data=df, x="transaction_date", y="house_price", color="#3
sns.regplot(ax=axes[0, 1], data=df, x='house_age', y='house_price', color="#361163")
sns.regplot(ax=axes[0, 2], data=df, x='mrt_distance', y='house_price', color="#36116
sns.regplot(ax=axes[1, 0], data=df, x='no_of_stores', y='house_price', color="#36116
sns.regplot(ax=axes[1, 1], data=df, x='latitude', y='house_price', color="#361163")
sns.regplot(ax=axes[1, 2], data=df, x='longitude', y='house_price', color="#361163")
```

Out[31]:

<AxesSubplot:xlabel='longitude', ylabel='house_price'>

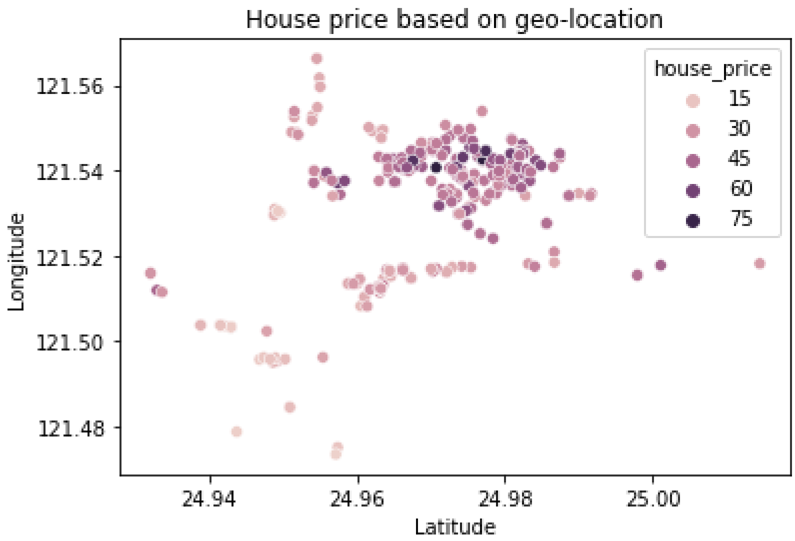


In [32]:

```
# Latitude, Longitude and House Price
sns.scatterplot(data=df, x="latitude", y="longitude", hue="house_price", sizes=(20,2
plt.xlabel("Latitude")
plt.ylabel("Longitude")
plt.title("House price based on geo-location")
```

Out[32]:

```
Text(0.5, 1.0, 'House price based on geo-location')
```




**Feature Selection**

We can see that the date of the house sale (transaction_date) has little to no correlation with the sale price (house_price). Therefore, I am removing this column.

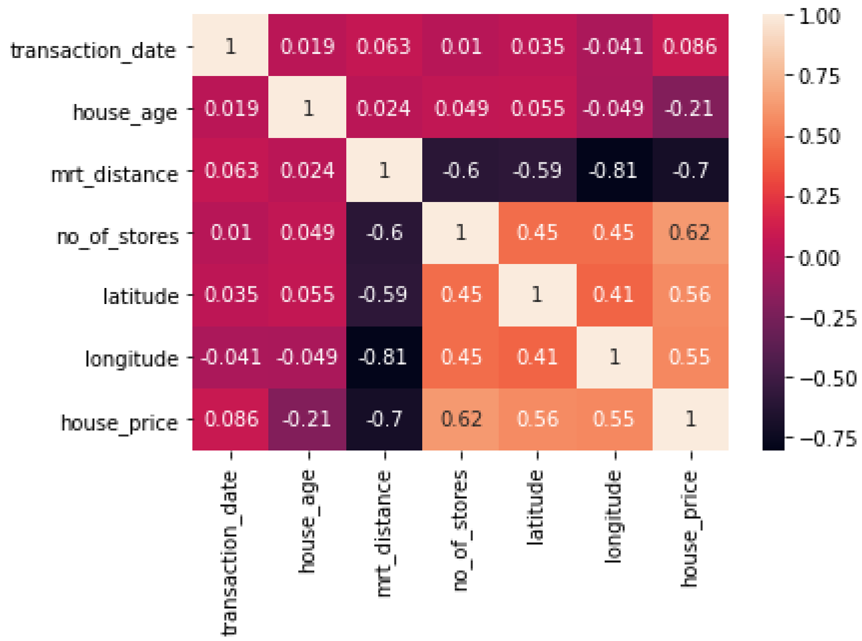
In [33]:

```
# Filter features by correlation
# What are the correlation of each attribute to the house price (house_price)?
print(abs(df.corr()['house_price']))

# Visualisation
sns.heatmap(df.corr(),annot=True)
```

```
transaction_date    0.086052
house_age           0.213846
mrt_distance        0.701939
no_of_stores        0.620631
latitude            0.564572
longitude           0.554680
house_price         1.000000
Name: house_price, dtype: float64
```

Out[33]:

<AxesSubplot:>




In [34]:

```
col_drop = df[['transaction_date']]
df.drop(columns = col_drop, inplace = True)
```

In [35]:

```
df.head()
```

Out[35]:

| | house_age | mrt_distance | no_of_stores | latitude | longitude | house_price |
|---|---|---|---|---|---|---|
| **0** | 32.0 | 84.87882 | 10 | 24.98298 | 121.54024 | 37.9 |
| **1** | 19.5 | 306.59470 | 9 | 24.98034 | 121.53951 | 42.2 |
| **2** | 13.3 | 561.98450 | 5 | 24.98746 | 121.54391 | 47.3 |
| **3** | 13.3 | 561.98450 | 5 | 24.98746 | 121.54391 | 54.8 |
| **4** | 5.0 | 390.56840 | 5 | 24.97937 | 121.54245 | 43.1 |

### Feature Scaling

Code adapted from: https://towardsdatascience.com/feature-scaling-effectively-choose-input-variables-based-on-distributions-3032207c921f (https://towardsdatascience.com/feature-scaling-effectively-choose-input-variables-based-on-distributions-3032207c921f) https://www.baeldung.com/cs/normalization-vs-standardization (https://www.baeldung.com/cs/normalization-vs-standardization)

In [36]:

```
# Import relevant package for feature scaling
from sklearn.preprocessing import MinMaxScaler
```

In [37]:

```
# Normalized dataset
min_max = MinMaxScaler()
df_norm = min_max.fit_transform(df)
df_norm = pd.DataFrame(df_norm, columns = df.columns)
df_norm.head()
```

Out[37]:

| | house_age | mrt_distance | no_of_stores | latitude | longitude | house_price |
|---|---|---|---|---|---|---|
| **0** | 0.730594 | 0.009513 | 1.0 | 0.616941 | 0.719323 | 0.397914 |
| **1** | 0.445205 | 0.043809 | 0.9 | 0.584949 | 0.711451 | 0.461997 |
| **2** | 0.303653 | 0.083315 | 0.5 | 0.671231 | 0.758896 | 0.538003 |
| **3** | 0.303653 | 0.083315 | 0.5 | 0.671231 | 0.758896 | 0.649776 |
| **4** | 0.114155 | 0.056799 | 0.5 | 0.573194 | 0.743153 | 0.475410 |

# Model Selection

In [38]:

```
# Import relevant packages for building the models
np.random.seed(42)
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.neighbors import KNeighborsRegressor

# Import relevant packages for the performance evaluation
from sklearn.metrics import r2_score, explained_variance_score, mean_absolute_error
from sklearn.metrics import mean_squared_error, median_absolute_error
# from sklearn.model_selection import cross_val_score ===== DELETE THIS
```

# Linear Regression

#### Fitting the model and making predictions

In [39]:

```
# Splitting features(X) and target(y)
X = df_norm.drop("house_price", axis = 1)
y = df_norm.house_price

# Splitting into training and testing sets
np.random.seed(42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)

# Build the Linear Regression Model on original dataset
lr = LinearRegression()

# Fit the Linear Regression Model
lr.fit(X_train, y_train)

# Using the model to make a prediction
y_pred_lr = lr.predict(X_test)
```

In [40]:

```
# Checking the shapes of training and testing data split
print(X_train.shape, y_train.shape,
     X_test.shape, y_test.shape)
```

```
(329, 5) (329,) (83, 5) (83,)
```

# Performance Evaluation

In [41]:

```
print("The performance evaluation of the Linear Regression model with the original d

# Checking the R squared score of the model [Best score: 1]
print("R-squared score is:", round(r2_score(y_test, y_pred_lr),2))

# Checking the Mean Squared Error of the model (MSE) [Best Value: 0.0]
print("Mean squared error is:", round(mean_squared_error(y_test, y_pred_lr),2))

# Checking the Mean Absolute Error (MAE), which shows the average difference between
print("Mean absolute error is:", round(mean_absolute_error(y_test, y_pred_lr),2))

# ===== DELETE BELOW
# # Checking the Median Absoute Error (MAD) [Best value: 0.0]
# print("Median absolute error is:", median_absolute_error(y_test, y_pred_lin))

# # Checking the Explained Variance Score
# # (Best possible score: 1)
# print("Explained variance score is:", explained_variance_score(y_test, y_pred_lin)
```

```
The performance evaluation of the Linear Regression model with the ori
ginal dataset are as follows:

R-squared score is: 0.67
Mean squared error is: 0.01
Mean absolute error is: 0.08
```

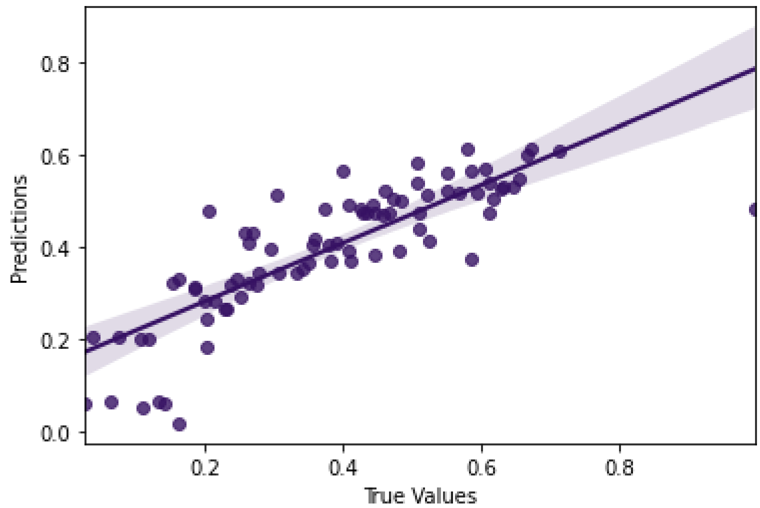
#### Visualizing the model

In [42]:

```
sns.regplot(x=y_test, y=y_pred_lr, color="#361163")
plt.xlabel("True Values")
plt.ylabel("Predictions")
```

Out[42]:

```
Text(0, 0.5, 'Predictions')
```




# Random Forest Regressor

#### Fitting a 'Random Forest Regressor' model

In [43]:

```
from sklearn.ensemble import RandomForestRegressor
```

In [44]:

```
# Splitting features(X) and target(y)
X = df_norm.drop("house_price", axis = 1)
y = df_norm.house_price

# Splitting into training and testing sets
np.random.seed(42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)

# Build the SVR Model
rf = RandomForestRegressor()

# Fit the model
rf.fit(X_train, y_train)

# Using the model to make a prediction
y_pred_rf = rf.predict(X_test)
```

# Performance Evaluation

In [45]:

```
print("The performance evaluation of the Random Forest Regression model are as follo

# Checking the R squared score of the model
# (Best score: 1)
print("R-squared score is:", round(r2_score(y_test, y_pred_rf),2))

# Checking the Mean Squared Error of the model (MSE)
# (Best Value: 0.0)
print("Mean squared error is:", round(mean_squared_error(y_test, y_pred_rf),2))

# Checking the Mean Absolute Error (MAE), which shows the average difference betweer
# Best value: 0.0
print("Mean absolute error is:",round(mean_absolute_error(y_test, y_pred_rf),2))
```

```
The performance evaluation of the Random Forest Regression model are a
s follows:

R-squared score is: 0.71
Mean squared error is: 0.01
Mean absolute error is: 0.07
```

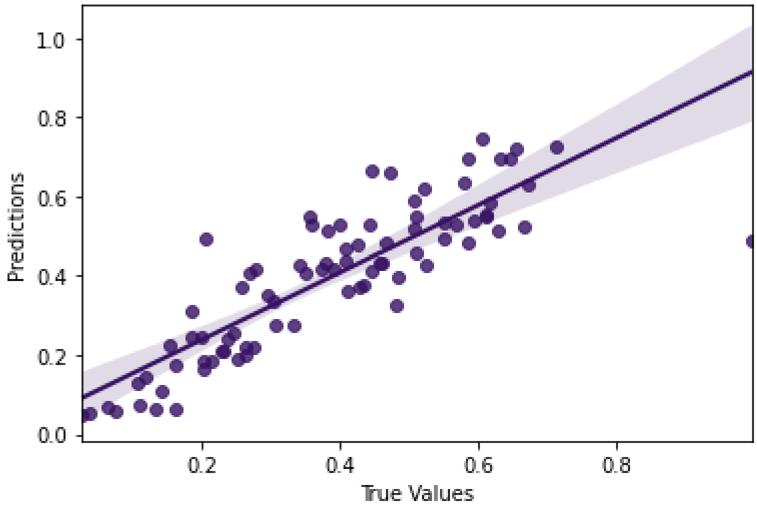
#### Visualize the model

In [46]:

```
sns.regplot(x=y_test, y=y_pred_rf, color="#361163")
plt.xlabel("True Values")
plt.ylabel("Predictions")
```

Out[46]:

```
Text(0, 0.5, 'Predictions')
```




# KNearestNeighbor

### Fitting a 'K-nearest Neighbor' model

In [48]:

```
# Splitting features(X) and target(y)
X = df_norm.drop("house_price", axis = 1)
y = df_norm.house_price

# Splitting into training and testing sets
np.random.seed(42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)

# Build the K-nearest Regressor model on original dataset
knr = KNeighborsRegressor(n_neighbors=5)

# Fit the K-nearest Regressor model
knr.fit(X_train, y_train)

# Using the model to make a prediction
y_pred_knr = knr.predict(X_test)
```

# Performance Evaluation

In [49]:

```
print("The performance evaluation of the K-nearest Regressor model are as follows:\r

# Checking the R squared score of the model
# (Best score: 1)
print("R-squared score is:", round(r2_score(y_test, y_pred_knr),2))

# Checking the Mean Squared Error of the model (MSE)
# (Best Value: 0.0)
print("Mean squared error is:", round(mean_squared_error(y_test, y_pred_knr),2))

# Checking the Mean Absolute Error (MAE), which shows the average difference betweer
# Best value: 0.0
print("Mean absolute error is:",round(mean_absolute_error(y_test, y_pred_knr),2))
```

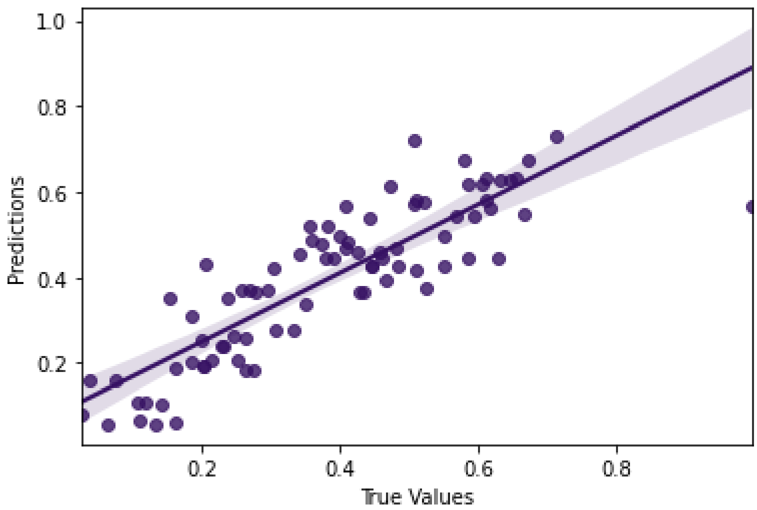
```
The performance evaluation of the K-nearest Regressor model are as fol
lows:

R-squared score is: 0.73
Mean squared error is: 0.01
Mean absolute error is: 0.07
```

In [50]:

```
sns.regplot(x=y_test, y=y_pred_knr, color="#361163")
plt.xlabel("True Values")
plt.ylabel("Predictions")
```

Out[50]:

```
Text(0, 0.5, 'Predictions')
```




# Improving the model

### K-Fold Cross Validation

*K-folds*

*Leave One Out*
Computationally expensive but suitable for small datasets

https://medium.com/analytics-vidhya/using-cross-validation-to-evaluate-different-models-regression-5f61ec89531 (https://medium.com/analytics-vidhya/using-cross-validation-to-evaluate-different-models-regression-5f61ec89531)

In [52]:

```
# Importing relevant package
# from sklearn.model_selection import LeaveOneOut
from sklearn.model_selection import cross_val_score, cross_val_predict
from sklearn import metrics

# Linear Regression
lr_cv = round(np.mean(cross_val_score(lr, X, y, cv=5))*100,2)
print(f"Accuracy of Linear Regression with Cross Validation: {lr_cv}%")

# Random Forest Regressor
rf_cv = round(np.mean(cross_val_score(rf, X, y, cv=5))*100,2)
print(f"Accuracy of Random Forest Regressor with Cross Validation: {rf_cv}%")

# K-Nearest Regresssor
knr_cv = round(np.mean(cross_val_score(knr, X, y, cv=5))*100,2)
print(f"Accuracy of K-Nearest Regressor with Cross Validation: {knr_cv} %")
```

```
Accuracy of Linear Regression with Cross Validation: 61.47%
Accuracy of Random Forest Regressor with Cross Validation: 73.69%
Accuracy of K-Nearest Regressor with Cross Validation: 71.27 %
```

In [ ]: