## **Project Report**

****

**Submitted by:**

**Submitted to:**

**Subject:**

**Section:**

**University of Management and Technology**

C-II Johar Town Lahore

**Predicting House Prices with Apache Spark**

Dataset:

In this we'll make use of the California Housing data set. These spatial data contain 20,640 observations on housing prices with 9 economic variables:

Longitude

Latitude

Housing median age

Total rooms

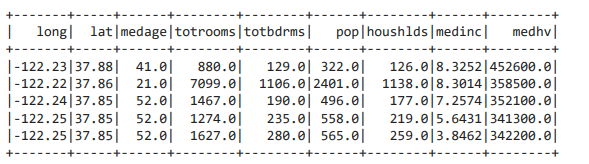
Total bedrooms

Population

Households

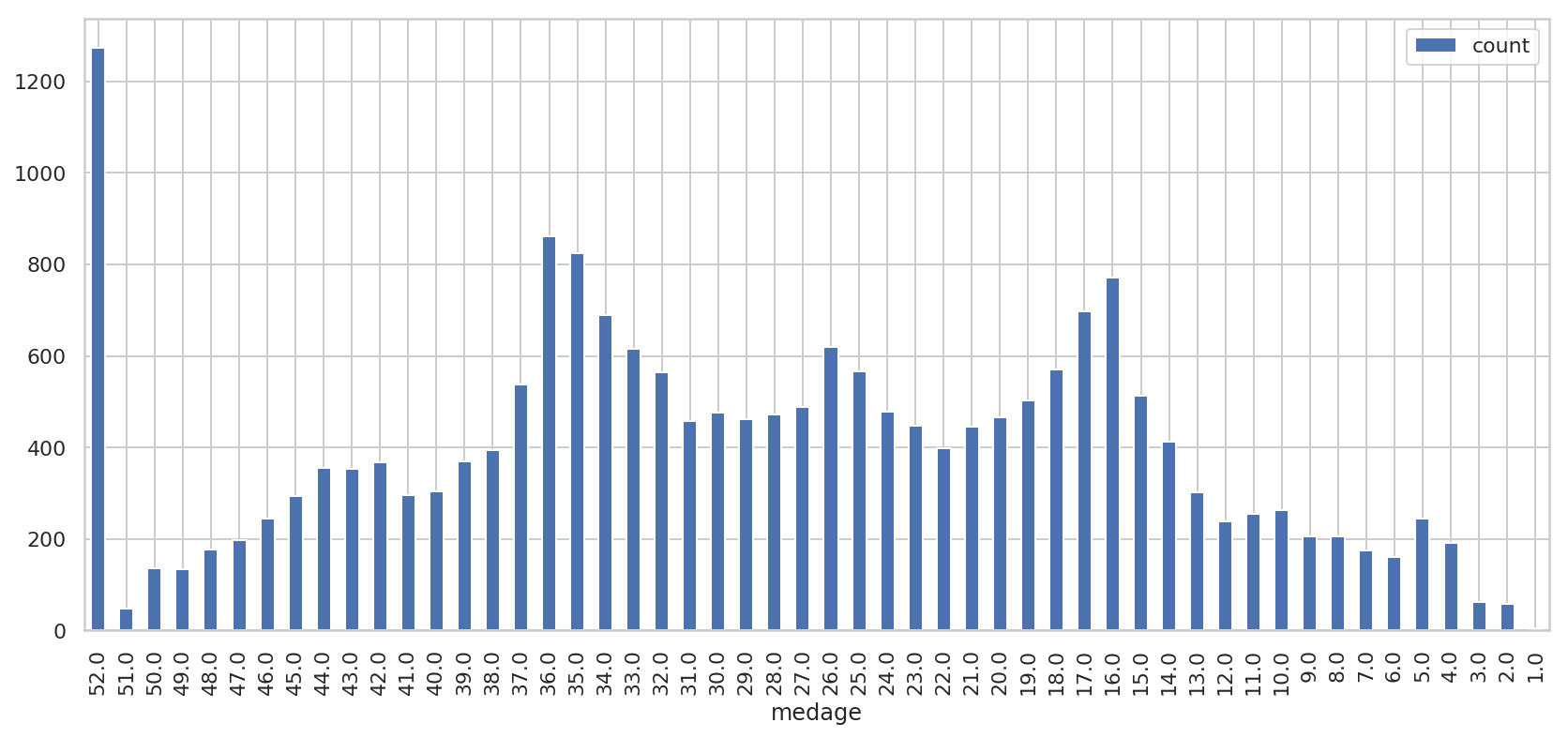
Median Income

Median house value

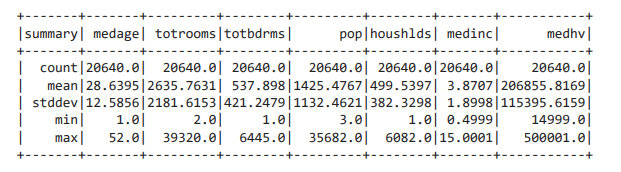


Data Exploration:

Distribution of the median age of the people living in the area



Most of the residents are either in their youth or they settle here during their senior years. Some data are showing median age < 10 which seems to be out of place.



Look at the minimum and maximum values of all the (numerical) attributes. We see that multiple attributes have a wide range of values: we will need to normalize your dataset.

Data Preprocessing:

With all this information that we gathered from our small exploratory data analysis, we know enough to preprocess our data to feed it to the model.

We shouldn't care about missing values; all zero values have been excluded from the data set.

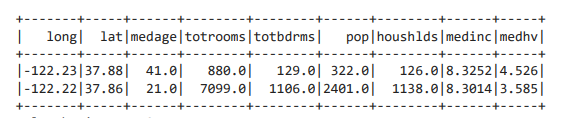
We should probably standardize our data, as we have seen that the range of minimum and maximum values is quite big.

There are possibly some additional attributes that we could add, such as a feature that registers the number of bedrooms per room or the rooms per household.

Our dependent variable is also quite big; to make our life easier, we'll have to adjust the values slightly.

Preprocessing the Target Values:

First, let's start with the medianHouseValue , our dependent variable. To facilitate our working with the target values, we will express the house values in units of 100,000. That means that a target such as 452600.000000 should become 4.526:



Feature Engineering:

Now that we have adjusted the values in medianHouseValue, we will now add the following columns to the data set:

Rooms per household which refers to the number of rooms in households per block group;

Population per household, which basically gives us an indication of how many people live in households per block group;

And Bedrooms per room which will give us an idea about how many rooms are bedrooms per block group;

Feature Extraction:

Now that we have re-ordered the data, we're ready to normalize the data. We will choose the features to be normalized.

featureCols = ["totbdrms", "pop", "houshlds", "medinc", "rmsperhh", "popperhh", "bdrmsperrm"]

Use a VectorAssembler to put features into a feature vector column:

Standardization:

Next, we can finally scale the data using StandardScaler. The input columns are the features , and the output column with the rescaled that will be included in the scaled\_df will be named "features\_scaled" :

Building a Machine Learning Model with Spark ML:

With all the preprocessing done, it's finally time to start building our Linear Regression model! Just like always, we first need to split the data into training and test sets. Luckily, this is no issue with the randomSplit() method:

# Split the data into train and test sets

train\_data, test\_data = scaled\_df.randomSplit([.8,.2], seed=rnd\_seed)

# Initialize `lr`

lr = (LinearRegression(featuresCol='features\_scaled', labelCol="medhv", predictionCol='predme maxIter=10, regParam=0.3, elasticNetParam=0.8, standardization= False))

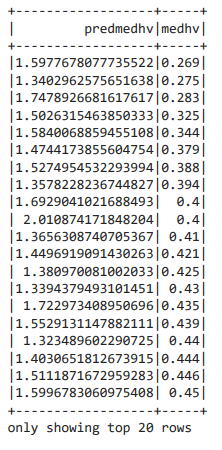
# Fit the data to the model

linearModel = lr.fit(train\_data)

Evaluating the Model:

With our model in place, we can generate predictions for our test data: use the transform() method to predict the labels for our test\_data . Then, we can use RDD operations to extract the predictions as well as the true labels from the DataFrame.

Generating Predictions:



Inspect the Metrics:

Looking at predicted values is one thing, but another and better thing is looking at some metrics to get a better idea of how good your model actually is.

Using the LinearRegressionModel.summary attribute:

Next, we can also use the summary attribute to pull up the rootMeanSquaredError and the r2

# Get the RMSE

print("RMSE: {0}".format(linearModel.summary.rootMeanSquaredError))

RMSE: 0.8729980899366503

print("MAE: {0}".format(linearModel.summary.meanAbsoluteError))

MAE: 0.6714989215155925

# Get the R2

print("R2: {0}".format(linearModel.summary.r2))

R2: 0.42213332730120356

The RMSE measures how much error there is between two datasets comparing a predicted value and an observed or known value. The smaller an RMSE value, the closer predicted and observed values are.

The R2 ("R squared") or the coefficient of determination is a measure that shows how close the data are to the fitted regression line. This score will always be between 0 and a 100% (or 0 to 1 in this case), where 0% indicates that the model explains none of the variability of the response data around its mean, and 100% indicates the opposite: it explains all the variability. That means that, in general, the higher the R-squared, the better the model fits our data.

**Conclusion:**

There's definitely some improvements needed to our model! If we want to continue with this model, we can play around with the parameters that we passed to your model, the variables that we included in your original Data Frame.