no	query name	Postgis	Turf js	Qgis
1.	Intersection	Returns a geometry representing the point-set intersection of two geometries. Ex: ST_Intersection('POINT(0 0)'::geometry, 'LINESTRING (2 0, 0 2)'::geometry) output: POINT(0 0)	Takes two polygon or multi-polygon geometries and finds their polygonal intersection. Ex: turf.intersect(poly1, poly2);	Extracts the portions of features from the input layer that overlap features in the overlay layer. 1.Using algorithm: processing.run("qgis:intersection", {parameter_dictionary }) 2.expression: intersection(geom1,geom)

2.	Line intersection	Returns a point geometry representing the line-set intersection geometries.
		Ex:
		ST_Intersection('LINESTRING (0 2 ,0 3)', 'LINESTRING (0 0, 0 2)')
		output: POINT(0 2)
			var line1 = turf.lineString([[126, -11], [129, -21]]);
			
			turf.lineIntersect(line1, line2);
				Creates point features where the lines from the two layers intersect.
				Example:processing.run("qgis:lineintersections", {parameter_dictionary })

3.	Area of Polygon	Returns the area of the surface if it is a polygon or multi-polygon. Ex :ST_Area(geom) For "geometry" type area is in SRID units. For "geography" area is in square meters.	Takes one or more features and returns their area in square meters. Ex:turf.area(polygon);	You can calculate the area directly from the attribute table. Create a new decimal/real field, and 1.use the $area expression.(area will be ellipsoidal)and use 2. area($geometry):(area will be planimetric)

4.	Azimuth	float ST_Azimuth(geometry origin, geometry target);
		
		SELECT degrees(ST_Azimuth(ST_Point(25, 45), ST_Point(75, 100))) AS degA_B.
		
		Returns the azimuth in radians of the target point from the origin point.
			Takes two points and finds the geographic bearing between them, i.e. the angle measured in degrees from the north line (0 degrees).
			Ex:
			point1 = turf.point([-75.343, 39.984]);
			
			turf.bearing(point1, point2);
				Returns the north-based azimuth as the angle in radians measured clockwise from the vertical on point_a to point_b.
				
				Ex:using Expression: degrees(azimuth(make_point(25, 45), make_point(75, 100)))

5.	Bounding box	An aggregate function that	Takes a set of features,	Layer can be vector either points,line or

no	query name	Postgis	Turf js	Qgis

		returns a box2d bounding box that bounds a set of geometries.
		
		Ex:
		
		SELECT ST_Extent(geom) as bbox
			calculates the bbox of all input features, and returns a bounding box.
			
			Ex:
			line = turf.lineString([[-74, 40], [-78, 42], [-82, 35]]);
			turf.bbox(line);
				polygon.
				Ex: 1. layer.extent()
				2.using Expression:
				bounds($geometry)

6.	Length	Returns the 2D length of the geometry if it is a LineString or MultiLineString.geometry are in units of spatial reference and geography are in meters (default spheroid) Ex: float ST_Length(geometry *a_2dlinestring*);	Takes a GeoJSON and measures its length in the specified units(default is kilometers)can be degrees, radians, miles, or kilometers. Ex: turf.length(line, {units: 'miles'});	Open Field Calculator from the attribute toolbar. Use Expression: $length

7.	bboxPolygon	Creates a rectangular Polygon from the minimum and maximum values for X and Y
		
		geometry ST_MakeEnvelope(float *xmin*, float *ymin*, float *xmax*, float *ymax*,
		integer *srid=unknown*);
		
		Ex:ST_MakeEnvelope(10, 10, 11, 11, 4326);
		
		output:
		POLYGON((10 10, 10 11, 11 11, 11 10, 10 10))
			Takes a bbox and returns an equivalent polygon.
			
			extent in minX, minY, maxX, maxY order.
			
			Ex:
			bbox = [0, 0, 10, 10];
			poly= turf.bboxPolygon(bbox);
				Using QuickWKT plugin .
				The bounding box would be treated as a polygon, and WKT would look like
				
				'POLYGON(xmin ymin,xmax ymax)'
				
				return bounding box polygon

8.	Centroid	Computes a point which is the geometric center of mass of a geometry.
		Ex:
		ST_Centroid('MULTIPOINT (1 0, -1 2, 1 1'));
		output:
		
		POINT(0.333333333333333 1)
			Takes one or more features and calculates the centroid using the mean of all vertices.
			
			Ex:
			
			turf.Polygon([[[-81, 41], [-88, 36], [-84, 31]]);
			
			turf.centroid(features);
				1.Using expression:
				
				EX:centroid($geometry)
				
				2.using algorithm

no	query name	Postgis	Turf js	Qgis
				Ex:processing.run("native:centroids", {params})

9.	Distance	For geom types returns the minimum 2D Cartesian distance between two geometries, in projected units. For geography types defaults to return the minimum geodesic distance between two geographies in meters.
		Ex:
		1.float ST_Distance(geometry *g1*, geometry *g2*);
		2.float ST_Distance(geography *geog1*, geography *geog2*, boolean *use_spheroid=true*);
			Calculates the distance between two point in degrees, radians, miles, or kilometers.
			from = turf.point([-75.343, 39.984]);
			to = turf.point([-75.534, 39.123]);
			options = {units: 'miles'};
			
			turf.distance(from, to, options);
				1.Use expression:
				Returns min distance (based on spatial ref) between two geometries in projected units.
				Distance(
				geom_from_wkt('POINT(4 4)'), geom_from_wkt('POINT(4 8)'))

10	Buffer	Computes a POLYGON or MULTIPOLYGON that represents all points whose distance from a geometry/geography is less than or equal to a given distance. geometry ST_Buffer(geometry *g1*, float *radius_of_buffer*, text *buffer_style_parameters* = ''); Ex: ST_Buffer(ST_GeomFromText('POINT(100 90)'), 50, 'quad_segs=8');	Calculates a buffer for input features for a given radius. Units supported are miles, kilometers, and degrees. Ex: turf.buffer(point, 500, {units: 'miles'});	1.Use Expression: buffer($geometry, radius) 2.use algorithm: 'native:buffer'
11	Difference	Returns a geometry representing the part of geometry A that does not intersect geometry B. This is equivalent to A – ST_Intersection(A,B). If A is completely contained in B then an empty atomic geometry of appropriate type is	Finds the difference between two polygons by clipping the second polygon from the first. Ex: turf.difference(polygon1, polygon2);	Returns a geometry that represents that part of geometry_a that does not intersect with geometry_b. 1.Use expression: difference(geometry_a, geometry_b).

no	query name	Postgis	Turf js	Qgis
		returned. ST_Difference('LINESTRING(50 100, 50 200)', 'LINESTRING(50 50, 50 150)')		2.use algorithm: 'native:diffrence'

12	Dissolve	To condense all the geometries in a table to one single geometry, you can pass an array to ST_Union:
		
		Ex:
		SELECT ST_Union(ARRAY(SELECT geom FROM tbl))
			Dissolves a FeatureCollection of polygon features, filtered by an optional property name:value. Note that mulitpolygon features within the collection are not supported.
			
			turf.featureCollection([turf.polygon([[[0, 0], [0, 1], [1, 1], [1, 0], [0, 0]]], {combine: 'yes'}),turf.polygon([[[0, -1], [0, 0], [1, 0], [1, -1], [0,-1]]], {combine: 'yes'}])
			
			Ex:
			
			turf.dissolve(features, {propertyName: 'combine'});
				This algorithm takes a vector layer and combines their features into new features.
				
				All output geometries will be converted to multi geometries.
				
				Ex:
				Use algorithm:
				'native:dissolve'

13	Union	Unions the input geometries, merging geometry to produce a result geometry with no overlaps. Ex: ST_Union('POINT(1 2), 'POINT(2 3)') output: MULTIPOINT(1 2 ,2 3)	Takes two multipolygon and returns a combined polygon. If the input polygons are not contiguous, this function returns a multipolygon feature. Ex: var union = turf.union(poly1, poly2);	Returns a geometry that represents the point set union of the geometries. 1.Using expression: geom_to_wkt(union(geom_from_wkt('POINT(4 4)'), geom_from_wkt('POINT(5 5)'))) 2.using algorithm: 'native:union'
14	Combine	Collects geometries into a geometry collection. The result is either a Multi* or a GeometryCollection. Ex: ST_Collect(ST_GeomFromText('	Combines a FeatureCollection of Point , LineString , or Polygon features into MultiPoint , MultiLineString , or MultiPolygon features.	Returns the combination of two geometries. 1.using Expression: geom_to_wkt(combine

no	query name	Postgis	Turf js	Qgis

		POINT(1 2)'), ST_GeomFromText('POINT(-2 3)'))
		
		output: MULTIPOINT((1 2),(-2 3))
			turf.featureCollection([turf.point([19.026432,47.49134]),turf.point([19.074497, 47.509548])]);
			
			Ex:
			
			turf.combine(fc);
				(geom_from_wkt('LINESTRING(3 3, 4 4, 5 5)'), geom_from_wkt('LINESTRING(3 3, 4 4, 2 1)'))) .
				
				output: MULTILINESTRING(….)
				
				2. using algorithm:
				
				‘native:collect’

15	Point Along Line	Returns a point interpolated along a line. Second argument is a float8 between 0 and 1 representing fraction of total length of linestring the point has to be located.
		
		select ST_LineInterpolatePoint('LINESTRING(25 50, 100 125, 150 190)', 0.20)
		
		output: POINT(51.5974135047432 76.5974135047432)
			Takes a Linestring and returns a point at a specified distance along the line.
			
			turf.lineString([[-83, 30], [-84, 36], [-78, 41]]);
			
			turf.along(line, 200, {units: 'meters'});
				Returns the point interpolated by a specified distance along a linestring geometry.
				
				1.using Expression
				
				line_interpolate_point(geometry,distance)
				
				2.using algorithm:
				
				"native:interpolatepoint"

16	Nearest Point	Returns the 2-dimensional point on geom1 that is closest to geom2. This is the first point of the shortest line between the geometries .
		
		Geometry ST_ClosestPoint(geometry *geom1*, geometry *geom2*);
		
		Ex:
		
		ST_ClosestPoint('POINT (160 40)','LINESTRING (10 30, 50 50, 30 110, 70 90, 180 140, 130 190)'))
		
		output: POINT(160 40)
			Takes a reference point and a FeatureCollection of Features with Point geometries and returns the point from the FeatureCollection closest to the reference. This calculation is geodesic.
			Ex:
			targetPoint = turf.point([28.965797, 41.010086], {"marker-color": "#0F0"});
			points = turf.featureCollection([turf.point([28.973865, 41.011122]),turf.point([28.948459, 41.024204])]);
			turf.nearestPoint(targetPoint, points);
				Returns the point on geometry1 that is closest to geometry2.
				
				1.using expression:
				
				closest_point(geometry1,geometry2)

17	Convex hull	Computes the convex hull of a	Takes a feature or	Returns the convex hull

no	query name	Postgis	Turf js	Qgis

		geometry.
		The convex hull is the smallest convex geometry that encloses all geometries in the input.
		Ex:
		ST_ConvexHull(ST_Collect(geom))
			a featurecollection and returns a convex hull polygon.
			
			Ex:
			turf.featureCollection([turf.point([10.195312, 43.755225])
			
			var hull = turf.convex(points);
				of a geometry.
				It represents the minimum convex geometry that encloses all geometries within the set.
				1.using expression:
				convex_hull(geom)
				2.using algorithm;
				"native:convexhull"

18	Concave hull	A concave hull of a geometry is a possibly concave geometry that encloses the vertices of the input geometry.
		The polygon will not contain holes unless the optional param_allow_holes argument is specified as true.
		geometry ST_ConcaveHull(geometry *param_geom*, float *param_pctconvex*, boolean *param_allow_holes = false*);
		Ex:
		ST_ConcaveHull(ST_Collect(geom), 0.80)
			Takes a set of points and returns a concave hull Polygon or MultiPolygon. Internally, this uses turf-tin to generate geometries.
			
			turf.featureCollection([turf.point([-63.601226, 44.642643]), turf.point([-63.591442, 44.651436]);
			turf.concave(points, {units: 'miles', maxEdge: 1});
				Computes the concave hull of the features in an input point layer.
				Ex:
				1.using algorithm:
				Computes the concave hull of the features in an input point layer.

19	Simplify	Returns a "simplified" version of the given geometry using the Douglas-Peucker algorithm. Will actually do something only with (multi)lines and (multi)polygons but you can safely call it with any kind of geometry.
		
		geometry ST_Simplify(geometry *geomA*, float *tolerance*);
		
		Ex:
		ST_Simplify(geom,0.1)
			Takes a geojson object and returns a simplified version. Internally uses simplify-js to perform simplification using the Ramer-Douglas-Peucker algorithm.
			
			Ex:
			turf.polygon([[[-70.603637, -33.399918], [-70.614624, -33.395332], [-70.639343, -33.392466], [-70.659942, -33.394759]]);
			
			turf.simplify(geojson, {tolerance: 0.01, highQuality: false});
				Simplifies a geometry by removing nodes using a distance based threshold.
				The algorithm preserves large deviations in geometries and reduces the number of vertices in nearly straight segments.
				1.using Expression:
				simplify(geometry,toler

no	query name	Postgis	Turf js	Qgis
				ance) 2.using Algorithm: "native:simplifygeometries
20	Offset line	Return an offset line at a given distance and side from an input line. Useful for computing parallel lines about a center line. For positive distance the offset is on the left side of the input line and retains the same direction. For a negative distance it is on the right side and in the opposite direction. Units of distance are measured in units of the spatial reference system. geometry ST_OffsetCurve(geometry *line*, float *signed_distance*, text *style_parameters*=''); Ex: ST_OffsetCurve(ST_GeomFromText('LINESTRING(164 16,144 16,124 16,104 16,84 16,64 16, 44 16,24 16,20 16,18 16,17 17)'), -15, 'quad_segs=4 join=round');	Takes a line and returns a line at offset by the specified distance. Ex: turf.lineString([[-83, 30], [-84, 36], [-78, 41]]) turf.lineOffset(line, 2, {units: 'miles'}); where, input can be GeoJSON. units can be can be degrees, radians, miles, kilometers, inches, yards, meters. distance to offset the line (can be of negative value)	Returns a geometry formed by offsetting a linestring geometry to the side. Distances are in the Spatial Reference System of this geometry. 1.Using expression: offset_curve(geometry, distance[,segments=8][,join=1][,miter_limit=2.0]) [ ] marks optional components. Ex: offset_curve($geometry, 10.5) I.e. line offset to the left by 10.5 units 2.using algorithm: native:offsetline
21	smooth	Returns a "smoothed" version of the given geometry using the Chaikin algorithm. For each iteration the number of vertex points will double. The function puts new vertex points at 1/4 of the line before and after each point and removes the original point.	Smooths a polygon or multipolygon. Based on Chaikin's algorithm . Warning: may create degenerate polygons. EX: turf.polygon([[[11, 0], [22, 4],	Smooths a geometry by adding extra nodes which round off corners in the geometry. 1.using expression: smooth(geometry[,iterations=1][,offset=0.25][,min_length=-1]

no	query name	Postgis	Turf js	Qgis

		geometry ST_ChaikinSmoothing (geometry *geom*, integer *nIterations = 1*, boolean *preserveEndPoints = false*);
		
		Ex:
		
		ST_ChaikinSmoothing(geom)
			[31, 0], [31, 11], [21, 15], [11, 11], [11, 0]]])
			
			turf.polygonSmooth(polygon, {iterations: 3})
				[,max_angle=180])
				
				Ex:
				
				geom_to_wkt(smooth(geometry:=geom_from_wkt('LineString(0 0, 5 0, 5 5)'),iterations:=1,offset:=0.2,min_length:=-1,max_angle:=180))
				
				output:LineString (0 0, 4 0, 5 1, 5 5)
				
				2.using algorithm:
				
				native:smoothgeometry

22	Polygonize	Creates a GeometryCollection containing the polygons formed by the constituent linework of a set of geometries.
		Input linework must be correctly noded for this function to work properly.
		Ex:
		geometry ST_Polygonize(geometry set *geomfield*);
		geometry ST_Polygonize(geometry[] *geom_array*);
			Polygonizes ((multi)linestring) into polygons
			
			Lines in order to polygonize.
				Creates a polygon layer whose features boundaries are generated from a line layer of **closed** features.
				
				The line layer must have closed shapes in order to be transformed into a polygon.
				
				Input layer : line
				
				1.using algorithm: native:polygonize

23	Polygon to line	Returns a LINESTRING representing the exterior ring (shell) of a POLYGON. Returns NULL if the geometry is not a polygon. geometry ST_ExteriorRing(geometry *a_polygon*); Ex: ST_ExteriorRing(	Converts a Polygon to (Multi)LineString or MultiPolygon to a FeatureCollection of (Multi)LineString. Ex: turf.polygon([[[125, -30], [145, -30], [145, -20], [125, -20], [125, -30]]]);	Converts polygons to lines. 1.using algorithm: native:polygonstolines where layer :polygon vector

no	query name	Postgis	Turf js	Qgis
		ST_GeomFromEWKT('POLYGON((0 0 , 1 1 , 1 2 , 1 1 , 0 0))')	turf.polygonToLine(poly);	
24	Clean Coordinates	Returns a version of the given geometry with duplicate consecutive points removed. The function processes only (Multi)LineStrings, (Multi)Polygons and MultiPoints but it can be called with any kind of geometry. If the tolerance parameter is provided, vertices within the tolerance distance of one another are considered to be duplicates. geometry ST_RemoveRepeatedPoints(geometry *geom*, float8 *tolerance*); Ex: ST_RemoveRepeatedPoints('MULTIPOINT ((1 1), (2 2), (3 3), (2 2))') output: MULTIPOINT(1 1,2 2,3 3)	Removes redundant coordinates from any GeoJSON Geometry. Ex: var multiPoint = turf.multiPoint([[0, 0], [0, 0], [2, 2]]); turf.cleanCoords(multiPoint).geometry.coordinates;	This algorithm finds duplicated geometries and removes them. 1.using algorithm: Ex: native:deleteduplicategeometries
25	Flip coordinates	Returns a version of the given geometry with X and Y axis flipped. Useful for fixing geometries which contain coordinates expressed as latitude/longitude (Y,X). geometry ST_FlipCoordinates(geometry *geom*); Ex: ST_FlipCoordinates(GeomFromEWKT('POINT(1 2)')) output:POINT(2 1)	Takes input features and flips all of their coordinates from [x, y] to [y, x]. Ex: turf.point([20.566406, 43.421008]); turf.flip(serbia);	Returns a copy of the geometry with the x and y coordinates swapped. Useful for repairing geometries which have had their latitude and longitude values reversed. Ex: 1.using expression: flip_coordinates(geometry) geom_to_wkt(flip_coordinates(make_point(1,

no	query name	Postgis	Turf js	Qgis
				2))) 2.using algorithm: native:swapxy

26	Kmean clustering	Returns K-means cluster number for each input geometry.
		The distance used for clustering is the distance between the centroids for 2D geometries, and distance between bounding box centers for 3D geometries.
		
		integer ST_ClusterKMeans(geometry winset *geom*, integer *number_of_clusters*, float *max_radius*);
		
		Ex:
		
		SELECT ST_ClusterKMeans(geom, 3) OVER() AS cid, parcel_id, geom
		FROM parcels;
			Takes a set of points and partition them into clusters using the k-mean. It uses the k-means algorithm.
			Input can be point (multi).
			numberOfClusters=Math.sqrt(numberOfPoints/2)
			
			Ex:
			turf.randomPoint(100, {bbox: [0, 30, 20, 50]});
			
			options = {numberOfClusters: 7};
			
			turf.clustersKmeans(points, options);
				Calculates the 2D distance based k-means cluster number for each input feature.
				
				If input geometries are lines or polygons, the clustering is based on the centroid of the feature.
				
				1.using algorithm:
				
				native:kmeansclustering

27	Dbscan clustering	Returns cluster number for each input geometry, based on a 2D implementation of the Density-based spatial clustering of applications with noise (DBSCAN) algorithm.
		Unlike ST_ClusterKMeans, it does not require the number of clusters to be specified, but instead uses the desired distance (eps) and density (minpoints) parameters to construct each cluster.
		
		integer ST_ClusterDBSCAN(geometry winset *geom*, float8 *eps*, integer *minpoints*);
		
		Ex:
		
		select name, ST_ClusterDBSCAN(geom, eps := 50, minpoints := 2) over
			Takes a set of points and partition them into clusters according to DBSCAN data clustering algorithm.
			Maximum Distance between any point of the cluster to generate the clusters (kilometers only).
			Minimum number of points to generate a single cluster, points which do not meet this requirement will be classified as an 'edge' or 'noise'.
			Ex:
			turf.randomPoint(100, {bbox: [0, 30, 20, 50]});
			
			maxDistance = 100;
			turf.clustersDbscan(points, maxDistance);
				Clusters point features based on a 2D implementation of Density-based spatial clustering of applications with noise (DBSCAN) algorithm.
				
				The algorithm requires two parameters, a minimum cluster size (“minPts”), and the maximum distance allowed between clustered points (“eps”).
				
				1.using algorithm:
				
				native:dbscanclustering

no	query name	Postgis	Turf js	Qgis
		() AS cid FROM table		

28	Rotate	Rotates geometry rotRadians counter-clockwise about the origin point.
		The rotation origin can be specified either as a POINT geometry, or as x and y coordinates.
		If the origin is not specified, the geometry is rotated about POINT(0 0).
		geometry ST_Rotate(geometry *geomA*, float *rotRadians*);
		
		geometry ST_Rotate(geometry *geomA*, float *rotRadians*, float *x0*, float *y0*);
		
		geometry ST_Rotate(geometry *geomA*, float *rotRadians*, geometry *pointOrigin*);
		
		Ex:
		
		1.Rotate 30 degrees counter-clockwise at x=50, y=160
		
		ST_Rotate('LINESTRING (50 160, 50 50, 100 50)', pi()/6, 50, 160)
			Rotates any geojson Feature or Geometry of a specified angle, around its centroid or a given pivot point.
			
			Angle:of rotation in decimal degrees, positive clockwise.
			
			Pivot:point around which the rotation will be performed (coordinates).
			
			Ex:
			
			turf.polygon([[[0,29],[3.5,29],[2.5,32],[0,29]]]);
			
			options = {pivot: [0, 25]};
			turf.transformRotate(poly, 10, options);
				Returns a rotated version of a geometry.
				
				rotate(geometry,rotation[,center])
				
				1.using Expression:
				
				rotate($geometry, 45, make_point(4, 5))
				
				2.using algorithm:
				
				This algorithm rotates feature geometries, by the specified angle clockwise
				
				Optionally, the rotation can occur around a preset point. If not set the rotation occurs around each feature's centroid.
				
				Ex:
				
				native:rotatefeatures

29	BBox Clip	Clips a geometry by a 2D box in a fast and tolerant but possibly invalid way.
		
		Topologically invalid input geometries do not result in exceptions being thrown. The output geometry is not guaranteed to be valid (in particular, self-intersections for a polygon may be introduced).
		
		geometry ST_ClipByBox2D(geometry *geom*, box2d *box*);
			Takes a Feature and a bbox and clips the feature to the bbox using lineclip. May result in degenerate edges when clipping Polygons.
			feature to clip to the bbox fetaure can be line or polygon extent in minX, minY, maxX, maxY order.
			Ex:
			bbox = [0, 0, 10, 10];
			poly = turf.polygon([[[2, 2], [8, 4], [12, 8], [3, 7], [2, 2]]]);
			turf.bboxClip(poly, bbox);
				This algorithm creates a new vector layer that only contains features which fall within a specified extent.
				
				Any features which intersect the extent will be included.
				
				1.using algorithm:
				
				native:extractbyextent

no	query name	Postgis	Turf js	Qgis
		Ex: ST_ClipByBox2D(geom, ST_MakeEnvelope(0,0,10,10)) ;		

30	Geometry type	Returns the type of the geometry as a string. Eg: 'LINESTRING', 'POLYGON', 'MULTIPOINT', etc.
		`text GeometryType(`geometry *geomA*`);`
		Ex:
		`SELECT GeometryType(ST_GeomFromText('LINESTRING(77.29 29.07,77.42 29.26,77.27 29.31,77.29 29.07)'));`
		output:LINESTRING
			Get GeoJSON object's type, Geometry type is prioritize.
			Ex:
			point = { "type": "Feature", "properties": {}, "geometry": { "type": "Point", "coordinates": [110, 40] } }
			var geom = turf.getType(point)
				Using QGIS expression, it can occur that you want to identify the geometry-type of a layer (point, line or polygon Multiple or single part).
				Ex:
				`string_to_array(geom_to_wkt($geometry),' ')[0]`